Tampere University

Jingyi Bai

Camera-based Object Tracking for Outdoor Scenes

Faculty of Information Technology and Communication Sciences (ITC)
Master's thesis
May 2019

Abstract

Jingyi Bai: Camera-based Object Tracking for Outdoor Scenes
Master's thesis
Tampere University
Masters Degree Programme in Data Engineering and Machine Learning
May 2019

This paper describes the implementation of object tracking from self-collected dataset. In this work we collect and preprocess data, train the object detection models, implement and compare two multiple object trackers, and draw the object movement trajectory. The performance of the detection model is evaluated based on precision, recall, F1 score, and detection speed, and the performance of the tracker is evaluated by accuracy and completeness of tracking. The challenge in the project comes from the impact of the detection quality of detection models on the tracking capabilities of trackers.

The data is collected from campus scenes and trained on pre-trained models provided by TensorFlow Object Detection model zoo. Experiment shows that Single shot multi-box detector is faster but has lower detection accuracy, Faster region-based convolutional neural network with residual networks has higher accuracy and performed better on our dataset. Then, we implement two multiple object trackers based on Kalman Filter Tracking and OpenCV Centroid Tracking. Hungarian algorithm is used for both to associate detected object with its new locations in new frames. The trackers are tested on MOTChallenge dataset for comparison, and the result shows that the performance of tracker with Kalman Filter Tracking is better than tracker with OpenCV Centroid Tracking, which is able to deal with the disappearance problem of the object caused by occlusion or instability of detection model. Finally, we draw the object movement trajectory, which reflects the movement behavior of pedestrians, cyclists and vehicles.

We believe that this project has application prospects in many security fields such as intelligent scene monitoring, pedestrian trajectory prediction in future, but there is still much space for improvement since the limitation of amount of dataset and the great impact of detection quality on object tracking.

Keywords: deep learning, object detection, TensorFlow Object Detection API, multiple object tracking, Kalman Filter tracking.

CONTENTS

1. INTRODUCTION

With the maturity of digital image processing theory and the development of computer data processing ability, computer vision has been widely applied. Computer vision can be understood as the installation of visual devices on machines. It is a subject of teaching machines how to see the world. It enables the machine to have visual functions similar to human beings, which can detect, recognize and track objects, and obtain information from images or multi-dimensional data. It takes images or video sequence as input, aims to represent and understand the environment.

As a research hotspot in computer vision field, object tracking is widely used in motion analysis, behavior recognition, video surveillance, intelligent transportation, robot navigation and other research directions. The main task is to find the position and trajectory of the interested moving object in the video sequence. Tracking algorithms need to adapt to various illumination transformations, motion blurring and appearance changes. Object tracking is an ill posed problem, for example, while tracking a vehicle from the back, if it turns the direction and shows the side to the camera, the tracker will lost the trajectory. Since most of the object tracking models are based on the position and scale of the object provided by the first frame, there are too few reference samples to produce a good model. When the appearance of the object changes, it is very difficult to keep tracking.

Object detection is the basis of object tracking. A system which able to detect, classify and recognize the object can track the target continuously. Object detection needs to find the exact location of the object in a given image and classify its category correctly. It answers the problem that where the object is and what it is. Although the variation of the size, the uncertainty of the angle and position make object detection more difficult, the algorithms develop rapidly. From the traditional method *Bag-Of-Words model (BOW)*, *Histogram of Oriented Gradient (HOG)*[6], *Deformable Parts Model (DPM)*[11] to recent deep learning method *Convolutional Neural Network (CNN)*, the accuracy, real-time performance and classifier robustness of object detection model have been improved.

In this work, we focus on the detection and tracking of pedestrians, cyclists and

vehicles in the square environment. For object detection, we need to understand the performance of different fine-tuned models on the dataset; for object tracking, we analyze whether the tracker can track the interested target accurately and the completeness of tracking trajectory.

The structure of this thesis is as follows. Chapter 2 introduces the theoretical background of machine learning and deep learning which are required concepts to understand the thesis. Chapter 3 describes the concept of object detection and network structures we used in experiments. In chapter 4, we introduce core algorithms used to implement and optimize the object tracker and trajectory mapping function. Chapter 5 talked about the assessment criteria of object detection and multiple object tracking, the factors affecting the performance of detection model and tracker is discussed, and the calculation method for accuracy metrics are introduced here. In chapter 6, we introduce the processes of the experiment start from data preprocessing, and the challenges of the experiment are discussed. The experimental results and the performance of object detection models and trackers are discussed in chapter 7, where also compare and analyze the influencing factors of the results. Chapter 8 summarizes the process of implementation and analyzes the experimental findings.

2. METHODS

This chapter describes the theoretical background of machine learning and deep learning, includes the different types of machine learning and the basic concepts of neural networks.

2.1 Machine Learning

Machine learning is a method to achieve artificial intelligence that trains a machine how to learn autonomously. *Tom Mitchell* provides a definition for machine learning: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.[30] It means, machine learning is mainly to study how to improve the performance of specific algorithms in empirical learning.

In recent years, machine learning has developed rapidly and profoundly changed the way of human production. Most industries working with large amounts of data have applied machine learning to actual production and different fields, such as medical diagnostics, speech and handwriting recognition, securities market analysis, search engines, fraud detection and self-driving technology.

Machine learning is used to analyze patterns from examples of input data, with the intention to generalize to new data. It is mainly divided into 3 categories: supervised learning, unsupervised learning, semi-supervised learning.

2.1.1 Supervised learning

Supervised learning constructs prediction models based on both input and output data. It trains a model from labelled dataset, maps the input value to the desired output value, and predicts the result based on this model when new data arrives. The model learns by comparing the actual output with desired output to find errors. The more labelled input and desired output samples provided to the model, the higher accuracy the model can achieve to analyze new data.

Supervised learning can be divided into regression and classification problem:

- Regression problem focuses on continuous values. The goal is to find the best fit. One example in regression is to predict the temperature of tomorrow according to previous temperature. The temperature here is a number, which is obviously a continuous value. Linear regression, support vector regression[8] are classical regression algorithms.
- When predicted output values are discrete, it is called classification. The classification model is dedicated to finding decision boundaries, which can divide the input data into different classes. For example, if we classify the weather to several categories, the desired output could be sunny, rainy, or cloudy, etc, which are subsets of weather but not a continuous value. This is a typical classification problem. In computer vision, a picture is identified as one of a number of classes, such cat, dog, or other animals. Support vector machine[17], Naive Bayes classification[25], and nearest neighbor method[5] are classification algorithms.

2.1.2 Unsupervised learning

Unsupervised learning only analyzes and groups data based on input data, and no target output is provided. The input dataset has no historical labels, which means, there is no reference data for training. The algorithm sorts out unlabelled data and mines hidden patterns or intrinsic structures in the data.

Clustering is one of the most commonly used unsupervised learning techniques. Clustering groups data based on similarity, which tries to make the data in same class highly similar, and the similarity of data between different classes as low as possible. Analysis of user purchase patterns, image color segmentation are typical applications of unsupervised learning.[3]

Compared with supervised learning, unsupervised learning is closer to the machine self-learning. However, the results of unsupervised learning are often unpredictable. Association rules learning[42], K-means clustering algorithm are widely used clustering algorithms.

2.1.3 Semi-supervised learning

Labelling data is very expensive in many practical problems. Usually, there is only a small amount of labelled data, but large amount of unlabelled data is easy to

collect. This situation has led to the rapid development of semi-supervised learning which can utilize both labelled and unlabelled samples.

Semi-supervised learning is a combination of supervised learning and unsupervised learning, which uses a large amount of unlabelled data and a few labelled data together as input training data for pattern recognition. Although unlabelled data is allowed for input data, there are still a lot of restrictions: unlabelled data should has and only has a certain class from labelled data, all labels of labelled data should be correct, the number of samples in each class of unlabelled data should be almost the same, the distribution of unlabelled data should be similar to the labelled data, and so on. Strict data requirements lead to less use of semi-supervised learning in practical.

2.2 Deep Learning

Deep learning is a branch of machine learning. It automatically extracts features from large amount of data, which is the biggest difference from traditional pattern recognition methods.

Deep learning is inspired by the human visual system. The information processing of the human visual system is hierarchical. External information enters human eyes and passes through four regions of the cerebral cortex, each of which produces different features. The feature becomes more abstract when the layer increases, meanwhile, it is easier to be classified.

Deep learning simulates the neural network of the human brain to recognize and represent data. It has been proven to be very effective in a variety of tasks. Deep neural networks, convolutional neural networks and recurrent neural networks have been applied in the fields of computer vision, speech recognition, natural language processing, audio recognition and bioinformatics.

2.2.1 Neural Networks

Neuron is one of the structural and functional unit of the human nervous system. In artificial neural networks, the artificial nodes represents neurons in human nervous system, and they are joined together to form the artificial network. A neuron model contains input, output, and computational functions.

Figure 2.1 shows the structure of the neuron. Nodes $\mathbf{a}_1$ to $\mathbf{a}_n$ are the components of the input vector, $\mathbf{w}_1$ to $\mathbf{w}_n$ are the weights of a synapse of the neuron. After each

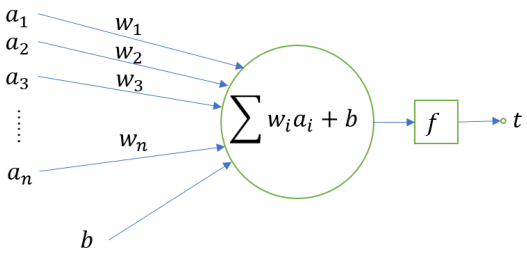


Figure 2.1 *A neuron contains multiple inputs, computational functions and one output.*

neuron accepts the input, it starts with a simple linear weighting, then all weighted nodes ($\mathbf{w} \times \mathbf{a}$) and offset b are summed up together before the activation function f is applied to perform nonlinear transformation and output $\mathbf{t}$. Different weights and activation functions lead to different outputs, which will be discussed later. A neural network training algorithm is to adjust the weight value, so that the prediction of the entire network becomes more accurate.

The neural network is connected by multiple neurons. Figure 2.2 shows a fully connected neural network, in which, each neuron in the *N*th layer is connected to all neurons in the *N-1*th layer, and the output of the *N-1*th layer neuron is the input of the neurons in *N*th layer.

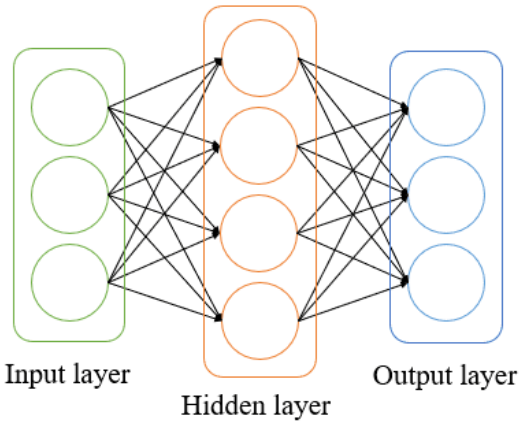


Figure 2.2 *Fully connected neural network has several layers, each node in each layer is connected with all nodes in next or previous layer.*

A common multilayer feedforward network consists of three parts:

- The leftmost layer is called the input layer, which has many neurons accept a large number of non-linear input messages;

- The rightmost layer is called the output layer, collect the result data after it has been transmitted, analyzed, and weighed in neuron links;
- The layers between the input layer and the output layer are called hidden layers, which are responsible for feature abstraction, and are invisible to the outside.

The number of nodes (neurons) in the hidden layer is variable, more nodes increase the nonlinearity of the network and lead to better robustness. The connection methods and the activation functions of neurons also influence the training effect of neural networks.

Activation function

The activation function introduces nonlinear factors to the neurons, which allows the neural network to approximate any nonlinear function arbitrarily, so that the neural network can be applied to many nonlinear models. Without activation function, output of each layer is a linear function from previous input, regardless of how many layers are connected in the network, the output is still a linear combination of inputs. Figure 2.3 shows common activation functions: *Sigmoid*, *Tanh* and *ReLu*.

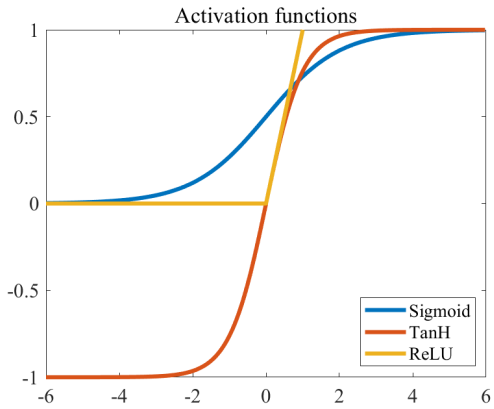


Figure 2.3 *Curve of Sigmoid, TanH and ReLU functions.*

Sigmoid function maps a real number to the interval of $(0, 1)$, so that the result can be interpreted as a class probability. It is commonly used in fully connected layer, works better in the situation when the difference of features is complicated or no big difference. *Sigmoid* is defined by the formula

$$sigmoid(x) = \frac{1}{1+e^{-x}}.$$

The value range for *Tanh* function is $[-1, 1]$. *Tanh* has good effect when the features are obviously different, which can highlight the features continuously in loop processes. The mean value of *Tanh* function is 0. *Tanh* is defined by the formula

$$tanh(x) = \frac{2}{1+e^{-2x}} - 1.$$

ReLu function is commonly used in deep learning models. When the input $signal < 0$, the output equals 0; when the input $signal \geq 0$, the output equals input. *Krizhevsky et al.* found that the stochastic gradient descent obtained by using *ReLu* will converge much faster than using *Sigmoid* and *Tanh*.[23] *ReLu* is defined by the formula

$$relu(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}.$$

Softmax function is another popular activation function which mainly used in multi-class neural network. It can compress a K-dimensional vector $\mathbf{z}$ with any real number into another K-dimensional vector $\sigma(\mathbf{z})$, so that the range of each element is between $(0, 1)$, and the sum of all elements is 1. *Softmax* is defined by the formula

$$\sigma_j(\mathbf{z}) = \frac{\exp(\mathbf{z}_j)}{\sum_{k=1}^{K} \exp(\mathbf{z}_k)}.$$

As shown in Figure 2.4, if value of vector $\mathbf{z}_1$ is larger than $\mathbf{z}_2, \mathbf{z}_3, \ldots, \mathbf{z}_n$, softmax maps the output value σ_1 close to 1, others are close to 0. Since the sum of all output elements is 1, it can be regarded as a probability value, then the task of multi-classification can be performed according to the probabilities.

Forward propagation and Backpropagation

Forward propagation and backpropagation are interdependent when training deep learning models.

Forward propagation calculates and stores the intermediate variables and output of the model along the input layer to the output layer. As shown in Figure 2.5, data goes from left to right, nodes i, j, k in previous layers are connected with the output

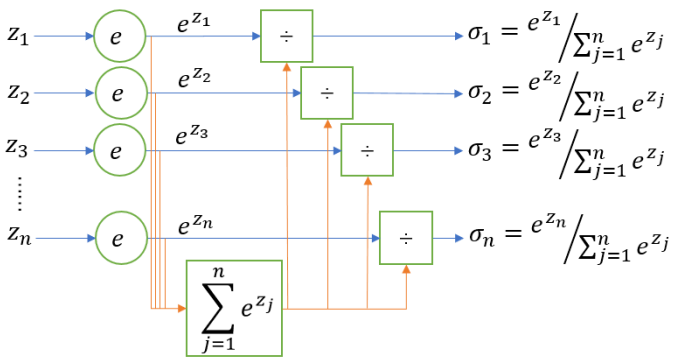


Figure 2.4 *Calculation process of Softmax function.[41]*

node l. To calculate the value of node l, first calculate the node j by summing up the weighted value of node i with the offset, and passing it through the activation function. Value of node k can be calculated according to node j in same way, and the value of node l depends on node k.

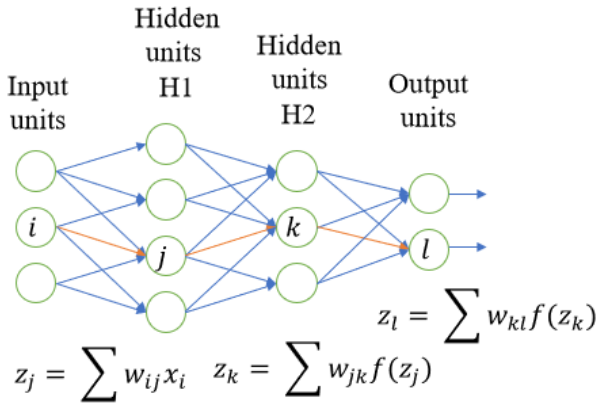


Figure 2.5 *Forward propagation process of neural network.*

The propagation process of forward propagation always contains errors. The neural network has multiple hidden layers, the neurons in each layer will produce predictions. Therefore, the prediction error needs to be considered and optimized for each layer. Backpropagation algorithm is designed to reduce this error.

Backpropagation is a method to calculate the gradient of neural network parameters. In general, backpropagation calculates the intermediate variables and the gradient of parameters from output layer to input layer.

In Figure 2.6, E is the total error of output l, the partial derivative of E for the output node l is calculated as $y_l - t_l$. t_l is the true value, $\frac{\partial y_l}{\partial z_l}$ refers to the activation

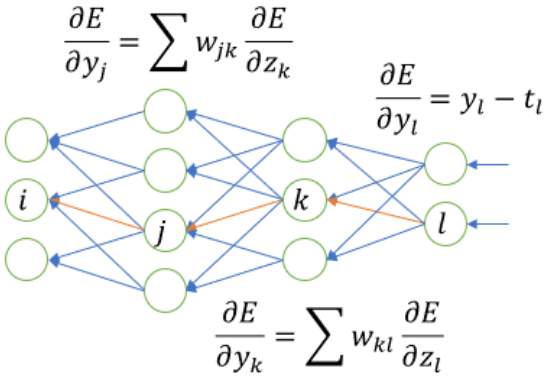


Figure 2.6 *Backpropagation process of neural network.*

function, and z_l is the weighted sum mentioned above ($w_{kl} \times y_k$). The node in previous layer is calculated in same way until the calculation is propagated to the input layer. Finally, the weight w_{ij} is calculated by $\frac{\partial z_j}{\partial x_i}$. To find the more suitable fit weights and output result, weights are adjusted continuously in process of forward propagation and back-propagation

2.2.2 Convolutional Neural Networks

Convolutional neural network (CNN) is a deep feedforward artificial neural network. It consists of one or more convolutional layers and a fully connected layer at the top, as well as weights and pooling layers. An image can be seen as one or more 2-dimensional vectors in image processing, and network structure enables the CNN to take advantage of this kind of input data.

Figure 2.7 shows the common structure of a CNN, which usually consists of convolutional layer, pooling layer, and fully connected layer. We will define each component in more detail in below.

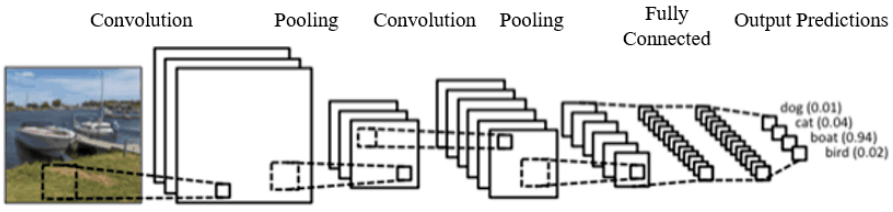


Figure 2.7 *Common structure of Convolutional Neural Networks.*

The traditional neural network is fully connected network, which has bad scalability and will calculate a huge amount of parameters, thus the network training is difficult

and time consuming. CNN avoids this problem by using sparse connectivity and shared weights. Neurons of CNN in hidden layer only connect to partial region of the previous layer. This region is called receptive field, which has the same size as the kernel in convolutional layer. Secondly, each convolution kernel can be seen as a feature extractor, which is independent of its position on image. That is, for the same convolution kernel, the features it extracts in one region can also be applied to other regions.Neurons belonging to the same feature map share the same weight parameter matrix. Sparse connectivity and shared weights greatly reduce the number of connection parameters needed to be trained, also reduce the complexity of network.

Convolutional layer

The role of the convolutional layer is to extract various features from the image. The convolutional layer is produced by calculating the inner product and sliding the convolution kernel on the previous input layer until all data is calculated.

Convolution is the result of summation after two variables multiplied in a certain rage. The convolution formula of sequence $x(n)$ and $h(n)$ is

$$x(n) * h(n) = \sum_{i=-\infty}^{\infty} x(i)h(n-i),$$

where $*$ represents convolution. When $n = 0$, sequence $h(-i)$ is the inversion of $h(i)$, which means $h(i)$ flips 180 degrees around the vertical axis. As n increases, $h(-i)$ is shifting. The sequence h is shifted and superimposed on sequence x to obtain the result sequence. It can be seen that convolution is a weighted superposition of a function (eg, unit response) on another function (eg, an input signal). At each position of the input signal, a unit response is superimposed to obtain an output signal.

The convolution dealing with sequence model is one-dimensional convolution, often used in the field of natural language processing. In the field of computer vision and image processing, we generally use two-dimensional convolution. As shown in Figure 2.8, one-dimensional convolution is the convolution of two sequences, two-dimensional convolution is the convolution of two matrices. The convolution formula of matrix $\mathbf{A}$ and $\mathbf{B}$ is

$$\mathbf{C}(x, y) = \sum_{i=-a}^{a} \sum_{j=-b}^{b} \mathbf{A}(i, j)\mathbf{B}(x-i, y-j),$$

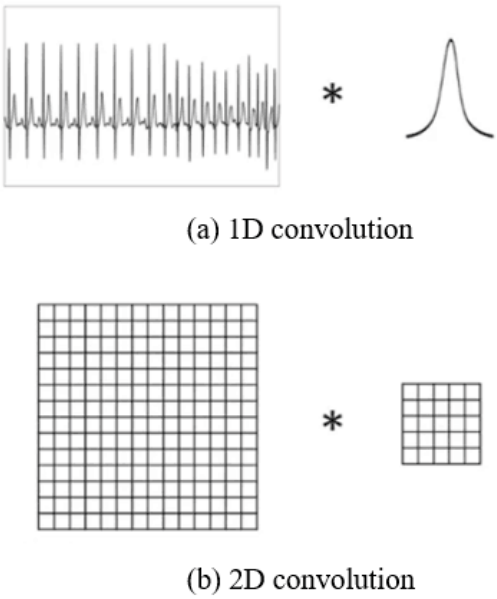


Figure 2.8 *(a) One-dimensional convolution is the convolution of two sequences (eg, input signal and unit response), (b) two-dimensional convolution is the convolution of two matrices (eg, original image and filter kernel).*

where a is the number of rows of $\mathbf{A}$, b is the number of columns of $\mathbf{A}$. Matrix $\mathbf{C}$ can be seen as the filtered image, $\mathbf{B}$ is the original image, and $\mathbf{A}$ is the filter kernel. Every element of the filter kernel is considered by $-a \leq i \leq a$ and $-b \leq j \leq b$.

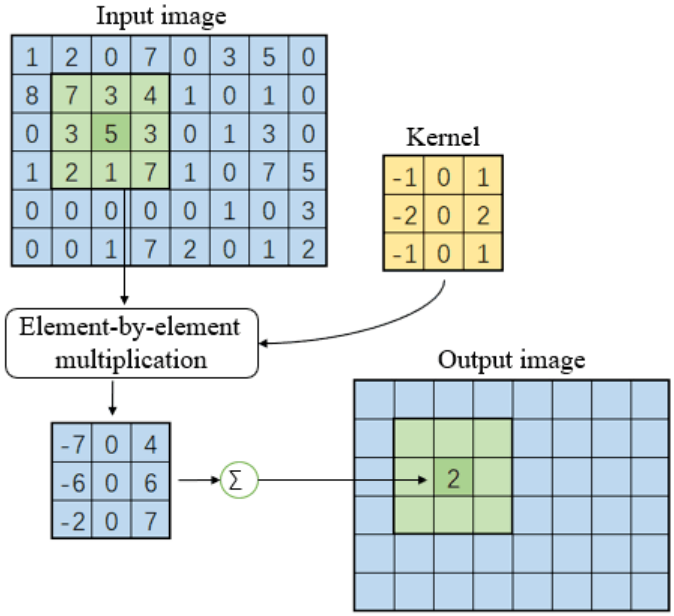


Figure 2.9 *Calculation process of 2D convolution.[?]*

As one-dimensional convolution, the essence is to flip the kernel matrix (rotate 180

degrees), then slide the kernel from top to bottom, from left to right, calculate the sum of the products of the intersection of the kernel and the original image. As shown in Figure 2.9, for each pixel of the image, the matrix which has the same size as kernel and takes the pixel as the center has been selected to calculate the product with the kernel matrix, then sum up all values in result matrix as the value of the pixel in same position.

Pooling layer

The features obtained from the convolutional layer can be used to train the classifier, but the large amount of calculation is a challenge. In order to further reduce the network parameters and avoid over-fitting, pooling method is processed on the convolutional layer.

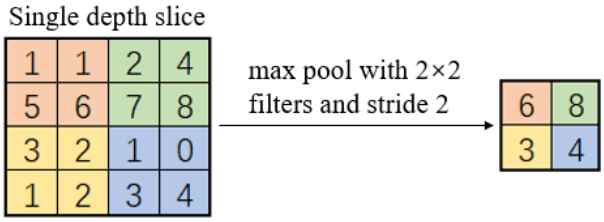


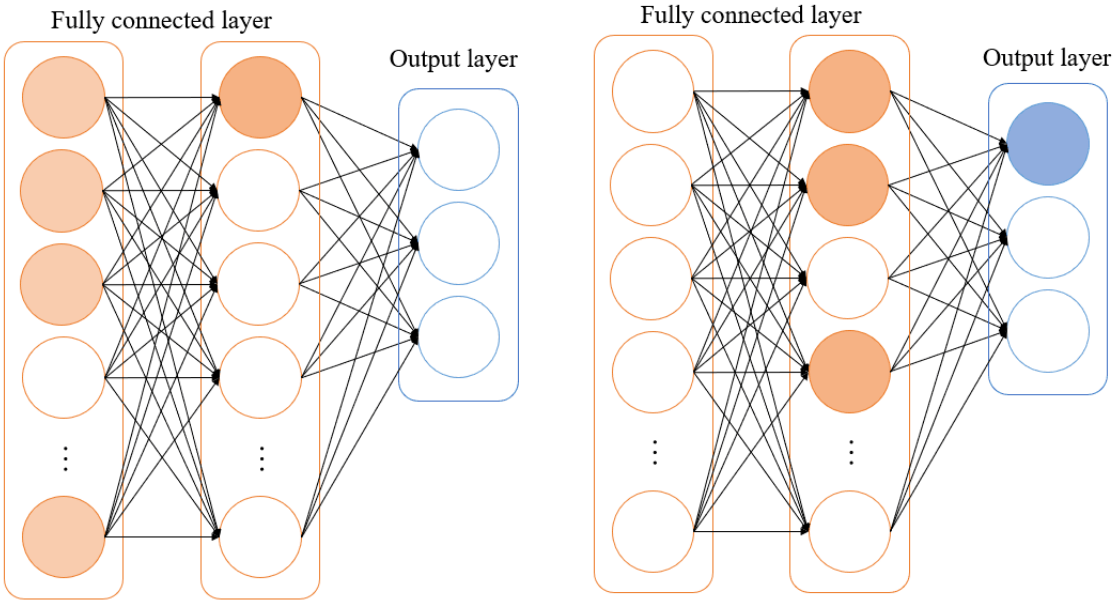
Figure 2.10 *Calculation process of Max pooling*

Figure 2.10 shows the concept of max pooling which selects the maximum value in the pooling window as the sample value. Mean pooling is also a common pooling method, which adds all the values in the pooling window and takes the average value as the sampled value. The role of the pooling layer is to reduce parameters and calculations, but maintain the main features. It reduces the sensitivity of the position and removes the redundant information in convolutional layer, improves the generalization ability of model.

Fully connected layer

Fully connected layer act as classifier throughout the CNN. The operation of convolutional layer and pooling layer is to map the original data to features in hidden layers, the fully connected layer plays the role of mapping the learned distributed feature representation to samples.

Each node of fully connected layer is connected to each node of the upper layer, which combines the output features of the previous layer. Figure 2.11 illustrates the classification process of fully connected layer.



(a) Sub-feature classification *(b) Classification*

Figure 2.11 *A fully connected layer combines sub-features extracted from convolutional layer and pooling layer into a main feature, then combines all activated main feature to obtain the classification.*

For example, assume that the task is to classify a cat, and the neural network model has been trained. From an image, we can determine whether the object is a cat by its head, tail, body, and fur, then the eyes, ears, mouth are the sub-features of the head. In sub-feature classification as shown in Figure 2.11(a), the nodes in first layer on left side indicate neurons with detail features extracted from model, which is sub-features mentioned above. And nodes in middle layer represent neurons with main features. When features are found, the neurons are activated. Activated neurons are represented as colored nodes, other neurons on the same layer has no obvious required features. By combining these sub-features, it can be found that they are the most likely to be a cat's head. So the neuron with feature *head* is activated. In Figure 2.11(b), when we combine these found main features, we find that cat is the most suitable result.

2.2.3 Commonly used convolutional networks

This section introduces three efficient CNNs: MobileNets, ResNet, Inception. They have their own advantages in different situations, and are used to provide lots of

pre-trained model for fine-tuning.

Fine-tuning

Good features can greatly improve the performance of the model, which requires large amount of labeled data. In fact, there is very little labeled data in real world, and most of the data needs to be collected and labeled manually, which costs time and resources. In this situation, the fine-tuning can be applied to reduce the amount of data required while maintaining the accuracy.

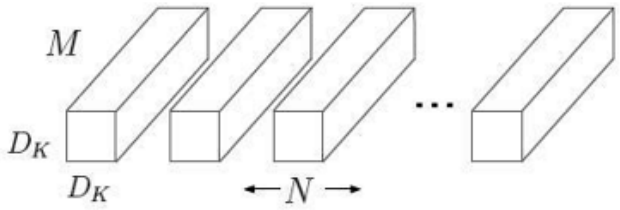
Fine-tuning is a common technique in transfer learning. Transfer learning is a method to transfer the learned knowledge from source dataset to target dataset. It transfers the well pre-trained parameters of the model to another model as the initial parameter value. The fine-tuned model is based on the model pre-trained on some large and classic dataset. It replicates all layers and parameters on the pre-trained model except the output layer, connects an self-created output layer which the number of output nodes equals to the number of classes. This fine-tuned model is then trained on target dataset.

Compared to the amount of data required to train a random initialized model, a fine-tuned model requires only a small amount of data to get good results. Although the pre-trained model may be trained on dataset without target objects, it still can extract general image features that can help to identify the edges, textures, and shapes, which is also effective for recognizing the desired target object. The fine-tuned model tends to achieve higher precision because of the better initial value of the parameters.

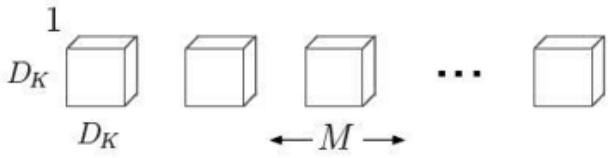
MobileNets

The model of a deep learning network requires a lot of memory and time to calculate, but applications in real-world scenarios usually require low latency and fast response, and devices has limited compute capability such complex models are difficult to be applied. So Google proposes a lightweight deep neural network called MobileNets[20], which is suitable for embedded devices such as mobile phones. The network is based on a streamlined architecture that uses deep separable convolution, which can effectively reduce network parameters. It focuses on optimizing the latency while taking the accuracy into account.

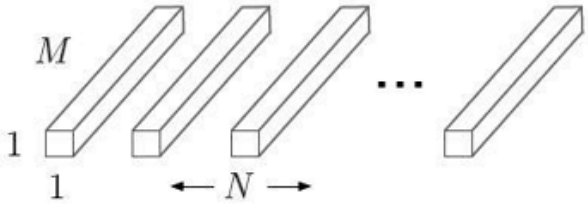
MobileNets decomposes standard convolution into depthwise convolution and pointwise convolution (1×1 convolution kernel) to reduce the model size and reduce the amount of calculation. The depthwise convolution goes through every channel, and the pointwise convolution is used to combine the output. Figure 2.12 illustrates how the standard convolution has been decomposed.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 2.12 *Decomposition of the standard convolution. Standard convolution is decomposed into depthwise and pointwise convolution to build a depthwise separatable filter.*[20]

MobileNets uses a large number of 3×3 convolution kernels, which greatly reduces the amount of calculation and latency. Although the accuracy has also decreased, but the overall performance is better than others. Light and efficient features make MobileNets ideal for mobile devices.

ResNet

The single-layer feedforward neural network is prone to over-fitting problems, so that the network structure has to be deepened. However, simply adding layers to increase

the depth of the network will cause the vanishing gradient problem, which makes the training of the deep network quite difficult. *C. Szegedy et al.* added auxiliary losses in the middle layer as additional supervision to deal with the problem, but there is no way to solve this problem completely at once.

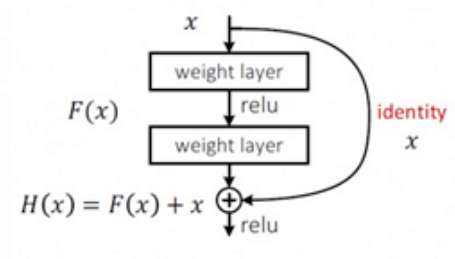


Figure 2.13 *Building block of residual learning.*

ResNet[16] uses the residuals to reconstruct the mapping of the network. The ResNet introduces the input to the result again, so that the input can propagate forward faster across the layer. The weight of the stacked layer tends to zero, which makes the learning easier. Figure 2.13 shows the residual block. $F(x)$ is a residual mapping with regard to identity x, $H(x)$ is the target mapping. According to residual function $F(x) = H(x) - x$, two weight layers are expected to fit $F(x)$ and let the $H(x)$ equals $F(x) + x$. ResNet learns residuals instead of features, uses residual equations instead of mapping inputs and outputs directly.

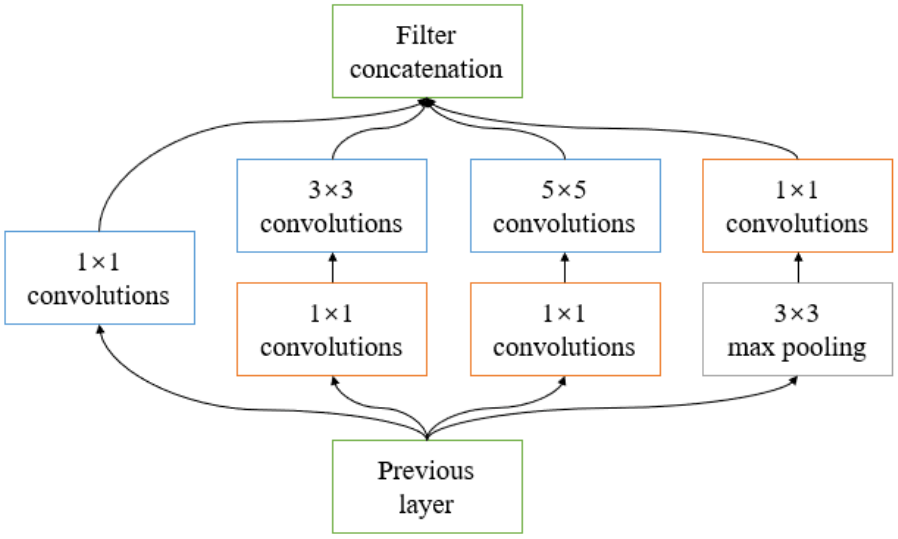


Figure 2.14 *Structure of Inception-v1.*

Inception

When the network layers becomes deeper(more layers) or wider(more neurons), the complexity of computation is higher and the model becomes difficult to optimize. Inception[37] is proposed to increase the depth and width of the network while reducing parameters. Figure 2.14 introduces the structure of Inception, which applies multi-layer parallel convolution kernels on same input. The 1×1 convolution kernel is added before 3×3 convolutions, 5×5 convolutions and after max pooling layer to reduce the thickness of the feature map. Each layer in the network can learn the sparse $(3 \times 3, 5 \times 5)$ feature or not sparse (1×1) feature, which increases the width and the scale adaptability of the network.

3. OBJECT DETECTION

How to parse the machine understandable information from the image is the central problem of computer vision. Figure 3.1 shows three major methods for a computer to understand a image: classification, localization and segmentation.

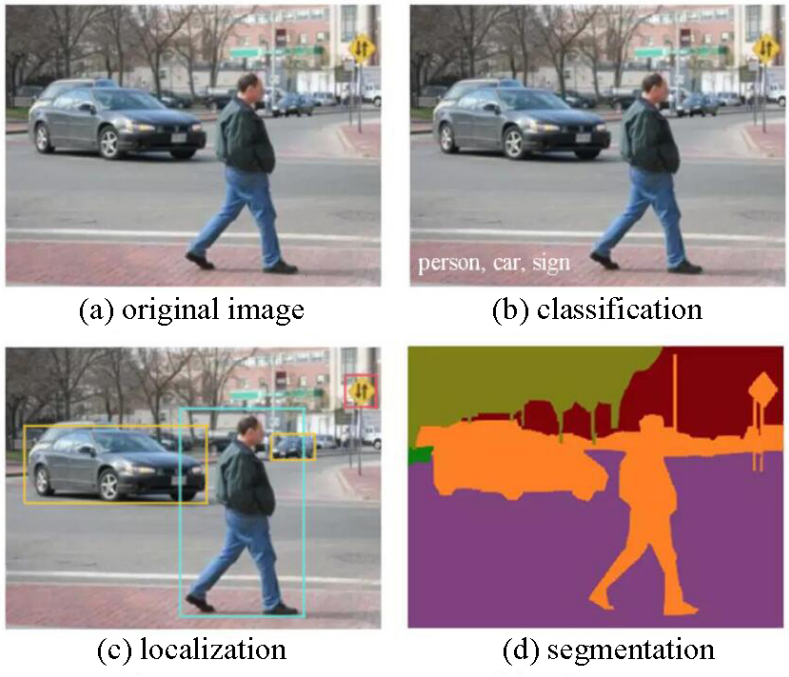


Figure 3.1 *Three major tasks for computer vision.[15]*

For an given image, image classification task is designed to find the category which image belongs, usually has a fixed set of labels. Localization task is based on classification, go further to find the position of the object in image, and the object is usually marked by bounding box. There is usually only one or fixed number of objects in object localization, and object detection is more general, which has uncertain number of objects in image. Segmentation is more advanced task of object detection. Object detection marks each object with bounding box, but segmentation requires to determine which pixels belong to which object.

We focus on object detection task in this work. In object detection, the image usually

contains more than one objects, and they are in different position and different classes. Object detection is proposed for this kind of problems, not only to analyze what is in the image, but also to identify where it is.

Figure 3.2 shows a classic object detection model which consists of feature extractor and classifier. Feature extractor provides representations of different levels of abstraction of the image, and classifier is responsible for class prediction and object localization according to the provided representations. The feature of input image is extracted by feature extractor first, then forwarded to the classifier as a reference for classification and localization. Finally, the classifier output the predicted class and position of the objects in image.

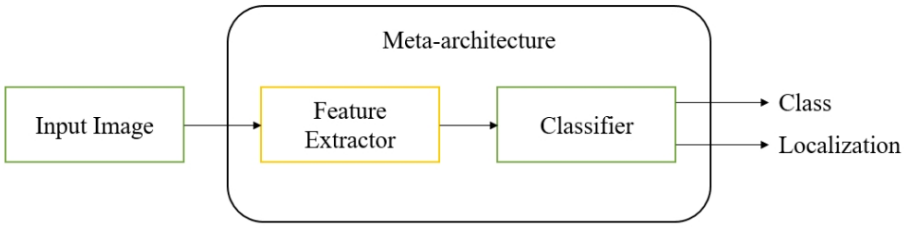


Figure 3.2 *Object detection model is composed of the feature extractor and classifier.[12]*

This is the classical approach before the CNN appears. For example, the HoG[6] is used as a feature extractor and SVM[17] is used as a classifier. With the development of CNN, the network structure applied to object detection becomes more complex, and it is difficult to clearly separate feature extractors and classifiers in recent networks. Although the main function of the network is still feature extraction and classification, it is usually divided by layers, and features are usually extracted multiple times.

As an important subset of computer vision, object detection technology is widely used in self-driving vehicles, smart cameras, face recognition and a large number of valuable applications.

Bounding box

In object detection, bounding box is used to describe the location of the target object. The bounding box is a rectangular box that can be represented by (x_1, y_1, x_2, y_2). (x_1, y_1) represents the upper left corner of the rectangle, and (x_2, y_2) represents the lower right corner.

Anchor box

To find the exact bounding box which describes the object correctly, the object detection algorithm will go through every pixels and sample large number of regions in the image. These regions are called anchor box.

(a) original image (b) anchor boxes (c) matched boxes

Figure 3.3 *(a) is the original image with grids. (b) shows the generated anchor boxes during prediction based on each grid. (c) shows the reserved boxes which matchs best.*

A common strategy to generate different shapes of anchor is that, take every pixel as centroid, and initialize a set of size $s_1, s_2, \ldots, s_n$, and a set of aspect ratio $r_1, r_2, \ldots, r_m$. The anchor will be collected if it fits s_1 or r_1. Figure 3.3 shows that a large number of anchor boxes are generated on the image first, then reserve only the best match one of each target object.

3.1 Region-based Convolutional Neural Networks

3.1.1 R-CNN

Regions with CNN features (R-CNN)[14] is the foundation of object detection based on convolutional neural networks. Figure 3.4 shows the core idea of R-CNN: selective-search method[38] extracts multiple region proposals from image, the CNN is applied to extract the feature for each region proposal, then SVM classifier is used to assign the labels and a bounding-box regression is used to adjust the border of region proposal and avoid multiple checkout boxes.

Although R-CNN has made great progress in object positioning, detection and classification compared with traditional CNN, there are still problems in real-time detection: the corresponding area of region proposals needs to be extracted in advance,

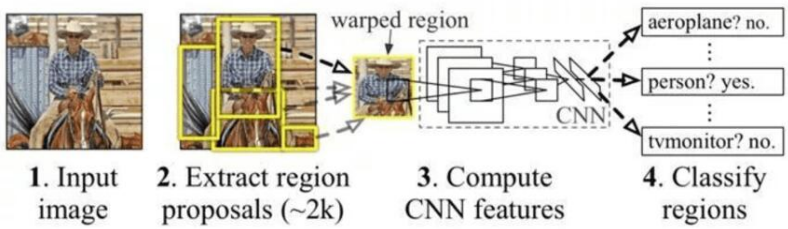


Figure 3.4 *Architecture of R-CNN.*

which occupies large disk space and takes a long time; the training is divided into 3 phases (proposal, classification, regression), and the model for each phase needs to be trained separately; the traditional CNN requires fixed-size input images, stretching and truncation (normalization) of input image leads to loss of information; every proposal region needs to be calculated by CNN, a large number of overlapping exists in thousands of regions, and the repeated feature extraction brings huge computational waste.

3.1.2 Fast R-CNN

In order to speed up the R-CNN system, Fast R-CNN[13] first extracts features from the whole input image and produce a conv feature map, then selects the Region of Interest from the feature map. It introduces a Region of Interest Pooling Layer to extract the same size output for each region proposal. RoI Pooling first divides the RoI into target number of meshes, then performs max pooling on each mesh to obtain the RoI feature vector with the same size. The feature vector is shared after the fully connected layer, then passed to classifier and regressor, so that the bounding box and label are output together. Figure 3.5

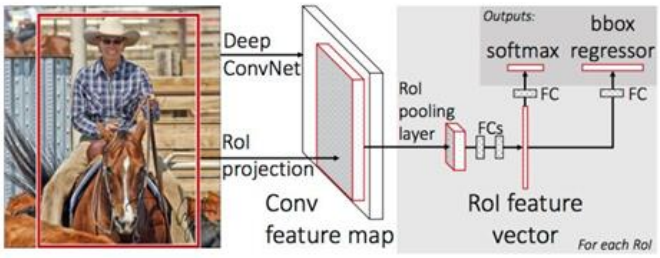


Figure 3.5 *Architecture of Fast R-CNN.*

3.1.3 Faster R-CNN

Fast R-CNN follows the selective search method of R-CNN to select regions, which is usually very slow. The main improvement made by Faster R-CNN[33] is to apply region proposal network (RPN) instead of selective search. Figure 3.6 shows the structure of the Faster R-CNN model, RPN is the only difference between Fast R-CNN and Faster R-CNN.

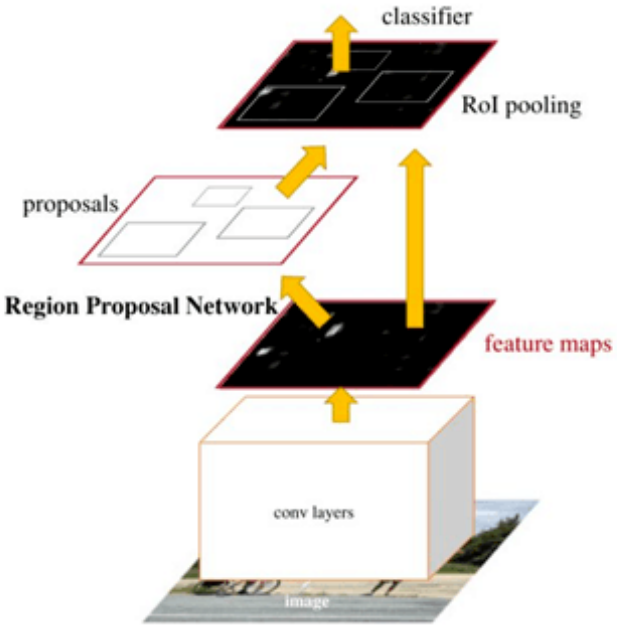


Figure 3.6 *Architecture of Faster R-CNN, which adds region proposal network from the Fast R-CNN.*

The RPN proposes pre-configured regions and detect if these regions are interest by using neural networks, and if so, predict a more accurate border. It can effectively reduce the cost of searching the borders in different shapes.

In Faster R-CNN, the anchor boxes are generated from a slide window as shown in Figure 3.7, and has been marked as positive or negative according to the ground truth. Multiple anchor boxes in different scale have been generated for each position. The input data for RPN is the coordinates of anchor box and its label of binary classification. RPN maps the data into scores (confidence of classification) and coordinates (bounding box). Score represents the probability that the object exists in the anchor box, and coordinates are used to define the position of the object. The region proposal obtained by the RPN is filtered based on the confidence and passed to the sub network of R-CNN for multi-classification and coordinate regression.

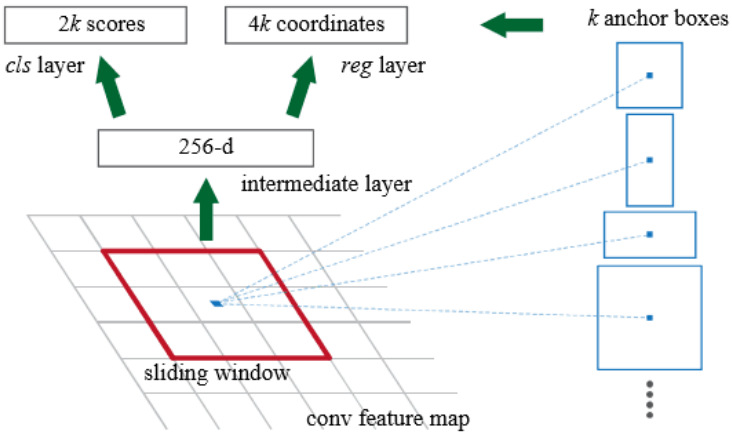


Figure 3.7 *Architecture of Region proposal network.*

3.2 Single Shot MultiBox Detector

Since the R-CNN algorithms are a bit slow for real-time detection, Single Shot MultiBox Detector(SSD)[27] is proposed. SSD is a simple and fast object detection model based on deep learning, has a good balance between accuracy and speed of calculation. It uses a single pass of a feedforward convolutional network, unifies regional proposals and classifications into one step. Compare to other networks, SSD has many prior boxes applied on feature maps in multiple different layers.

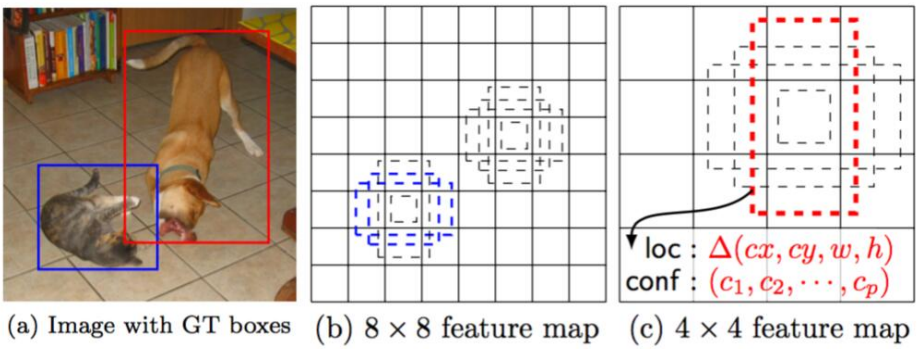


Figure 3.8 *Priors anchor for SSD.[27]*

SSD generates prior boxes with different scales in feature map in order to improve the detection accuracy. In training phase, the algorithm first matches the prior box and the ground truth box, predicts the bounding boxes based on the prior boxes, which reduces the training difficulty. Finally, predicts the object position by softmax classification and bounding box regression algorithms. In the prediction phase, predicts the offset of each prior box and the confidence for each class. The

prior box has multiple different scales in different feature maps, and has different aspect ratios in the same feature map, which can basically cover various shapes and sizes of object in the input image.

Suppose the size of feature map is $W \times H$, then there are $W \times H$ prior box centers generated and evenly distributed on the whole feature map. Multiple prior boxes with different aspect ratio are generated on each center. One ground truth can match to multiple prior boxes. Figure 3.8 shows the prior boxes for an example image generated by SSD. On 8×8 feature map in Figure 3.8 (b), there are 4 prior boxes generated on each center, which are more conducive to detect small-scale objects. The blue dotted boxes in Figure 3.8 (b) match the ground truth box of cat. The boxes extracted from Figure 3.8 (c) is more conducive to the detection of large-scale objects, so the red dotted box in Figure 3.8 (c) matches the ground truth box of dog. It can be seen that large-scale feature maps are used to detect small objects, while small-scale feature maps are used to detect large objects.

Figure 3.9 shows the structure of SSD, which adds several convolutional layers for feature extraction behind the original VGG16 network to predict the offset and confidence. The network combines the output of rest 6 convolutional layers to predict location and confidence. The SSD network extracts features and generates prior boxes from 6-layer feature maps. Multi-scale convolutional layer is designed to extract the features of object on different scales. The previous layer only detects smaller objects and next layer detects larger objects. This process is repeated multiple times to ensure that the object is detected at a variety of different scales.

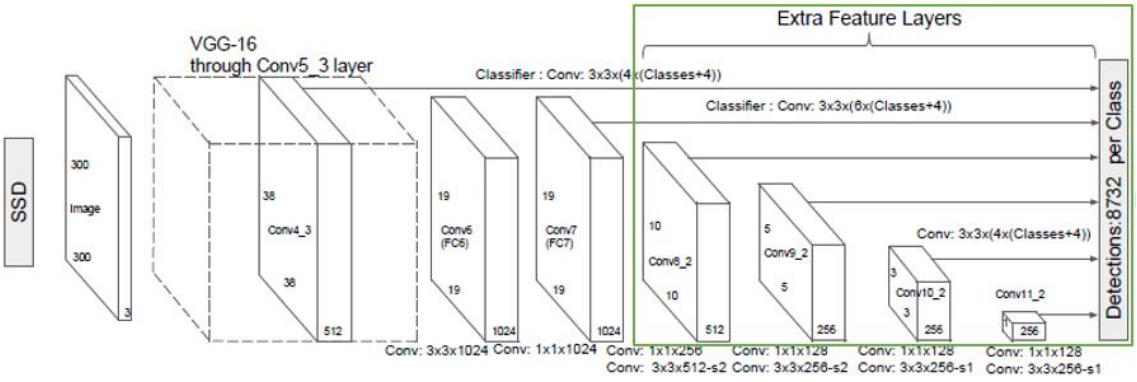


Figure 3.9 *Architecture of SSD. Convolutional layers in green box are the added feature extractor from the original VGG16 network.[27]*

4. MULTIPLE OBJECT TRACKING AND TRAJECTORY MAPPING

The video is composed of n consecutive frames, multiple targets continuously move, enter and exit from the first frame to the last frame. The main task of object tracking is to distinguish each target from other targets, locate these targets in each frame of given video, maintain their IDs and track their trajectories. Major challenges are occlusions (targets are occluded by the scene or other targets), similar appearance (separate different targets), complex motion (sharp turn or suddenly go back), false detection, etc.[28]

In general, the multiple object tracking algorithm is mainly divided into the following types:

- Point Tracking: the object is expressed as point. The tracking problem can be expressed by the relationship of points detected between frames. The challenge is occlusion, entry and exit screens, wrong detection problems.
- Kernel Tracking: focus on the shape and appearance of the object. The kernel function is associate with a histogram, the object is tracked by calculating the motion of the kernel.
- Silhouette Tracking: estimate the interested area in each frame. The silhouette is inferred from the given model.

In this work, the point tracking algorithm is used to track the objects. Since the detection of objects are provided by deep learning model, this chapter only focus on multiple object tracking and trajectory mapping. This chapter introduces the following three parts: the assignment algorithm, the object tracking algorithm and the tracking trajectory mapping algorithm. The details of the algorithms will be discussed in this chapter.

4.1 Hungarian algorithm

In order to track multiple objects simultaneously, each object detected in the video should be assigned a unique ID, then the trackers could distinguish objects based on IDs. The same object should be matched in continuous frames to maintain their ID. The challenge is how to find the bounding boxes in nth frame and $n+1$th frame which represent the same object. Since the short interval between each frame, the displacement of objects is small. Therefore, the bounding boxes in continuous frames with the smallest displacement distance can be regarded as the same. This kind of assignment problem can be solved with Hungarian algorithm.

The Hungarian algorithm[24] is developed by *Harold Kuhn* to solve assignment problems in polynomial time, and it is the most common algorithm to find the perfect match for the lowest cost. The core of the algorithm is to find the augmented path. The basic theoretical basis is that for the cost matrix, the optimal task assignment problem is unchanged if a value is added to or subtracted from any row or column.

The calculation steps of Hungarian algorithm are as follows:

1. Each row subtracts its smallest entry, then for the result matrix, each column subtracts its smallest entry.
2. Draw lines in rows and columns as few as possible to cover all 0 elements in the matrix.
3. If the minimum number of lines is less than the number of matrix rows, subtract the smallest entry of uncovered entries from each uncovered row and add it to each covered column.
4. Repeat all steps until the minimum number of lines equal to the number of matrix rows.

The follow example describes the step of Hungarian algorithm. Assume that n tasks is given to n workers, where a, b, c, d represent workers, and $1, 2, 3, 4$ represent tasks, a_3 represents the worker a is assigned to complete task 3. The $n \times n$ cost matrix is

$$\begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{bmatrix},$$

assume a_1, b_4, c_1, d_2 are the minimum values, and d_3 is the smallest entry in 3rd column. After subtracting the minimum value in each row and column, the matrix becomes

$$\begin{bmatrix} 0 & a_2' & a_3' & a_4' \\ b_1' & b_2' & b_3' & 0 \\ 0 & c_2' & c_3' & c_4' \\ d_1' & 0 & 0 & d_4' \end{bmatrix}.$$

To cover 0 elements as more as possible, the line is drawn on 1st column, 2nd row and 4th row. Rest elements are a_2', a_3', a_4', c_2', c_3', and c_4'. Assume that a_2' is the smallest entry not covered by any line, subtract it from each uncovered row and add it to each covered column, the matrix becomes

$$\begin{bmatrix} 0 & a_2' & a_3' & a_4' \\ b_1' & b_2' & b_3' & 0 \\ 0 & c_2' & c_3' & c_4' \\ d_1' & 0 & 0 & d_4' \end{bmatrix} \rightarrow \begin{bmatrix} -a_2' & 0 & a_3' & a_4' \\ b_1' & b_2' & b_3' & 0 \\ -a_2' & c_2' & c_3' & c_4' \\ d_1' & 0 & 0 & d_4' \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & a_3' & a_4' \\ b_1' & b_2' & b_3' & 0 \\ 0 & c_2' & c_3' & c_4' \\ d_1' & 0 & 0 & d_4' \end{bmatrix}.$$

Repeat the previous steps, 4 lines can be drawn to cover all 0 elements in current matrix. Assign 0 costs for each row, the matrix can be represented as

$$\begin{bmatrix} 0 & 0 & a_3' & a_4' \\ b_1' & b_2' & b_3' & 0 \\ 0 & c_2' & c_3' & c_4' \\ d_1' & 0 & 0 & d_4' \end{bmatrix}.$$

Now an optimal assignment of zeros is possible, the tasks can be assigned as $a \rightarrow 2, b \rightarrow 4, c \rightarrow 1, d \rightarrow 3$.

4.2 Kalman Filter

Once the objects are distinguished, each object is tracked separately. Due to the complexity of the scene, the object will be occluded by background or missed by unstable detector in few frames then appear again, it is necessary to predict the possible position of object in next frame. The prior knowledge tells us the moving track of the object is smooth, the current state of the object is related to the previous state, and the filtering method can take these prior knowledge into account to obtain a more accurate track. Thus, Kalman filter is used to achieve object tracking in this work.

Kalman filtering is a recursive estimation, that is, the estimated current state can

be calculate as long as the estimated previous state and the observed current state are known, and it is not necessary to record the observed or estimated historical information.[39]

For object tracking, the observed current position (observed/measured position) is able to be measured by some positioning techniques, and the estimated current position (predicted position) can be predicted by the position and speed of previous position according to the experience that the moving object is often moving in constant speed. Positioning result is the weighted average of the observed position and predicted position, the magnitude of the weight depends on the degree of uncertainty of both position. It can be proved mathematically that when the uncertainty of observed position and predicted position are both distributed as linear Gaussian distribution, weighting according to Kalman's method is optimal.[10]

The state difference equation for a linear system at time k is

$$\mathbf{x}_k = \mathbf{A}_k\mathbf{x}_{k-1} + \mathbf{B}_k\mathbf{u}_k + \mathbf{w}_k,$$

where $\mathbf{x}$ is the state vector of the system, $\mathbf{u}$ is the system input, $\mathbf{A}$ is a state transition matrix and $\mathbf{B}$ is a matrix that converts the input $\mathbf{u}$ to a state. The random variable $\mathbf{w}$ is the system noise.

Assume in real world, a uniformly accelerating moving object with mass m receives force f_t. The state vector of the system includes displacement and velocity, which can be expressed in $\mathbf{disp}_t$ and $\mathbf{vel}_t$ (derivatives of $\mathbf{disp}_t$). The formula for the displacement and velocity of this object from time $t-1$ to t is

$$\mathbf{disp}_t = \mathbf{disp}_{t-1} + (\mathbf{vel}_{t-1} \times \Delta t) + \frac{f_t(\Delta t)^2}{2m},$$

$$\mathbf{vel}_t = \mathbf{vel}_{t-1} + \frac{f_t \Delta t}{m}.$$

The state vector $\mathbf{x}$ of the system at time t is

$$\mathbf{x}_t = \begin{bmatrix} \mathbf{disp}_t \\ \mathbf{vel}_t \end{bmatrix},$$

and the input variable $\mathbf{u}$ is the acceleration of the object, which can be represented as

$$\mathbf{u}_t = \frac{f_t}{m},$$

so the equation for the state of the system is

$$\begin{bmatrix} \mathbf{disp}_t \\ \mathbf{vel}_t \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{disp}_{t-1} \\ \mathbf{vel}_{t-1} \end{bmatrix} + \begin{bmatrix} \frac{(\Delta t)^2}{2} \\ \Delta t \end{bmatrix} \frac{f_t}{m}.$$

Therefore,

$$\mathbf{A} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix},$$

$$\mathbf{B} = \begin{bmatrix} \frac{(\Delta t)^2}{2} \\ \Delta t \end{bmatrix}.$$

Although this model can fully estimate the system state and calculate both displacement and velocity, in practical, actual systems are complicated and difficult to create a model. Moreover, the error in the calculation is magnified by $\mathbf{A}$ times in the recursion, and the deviation of estimated value is large.

Therefore, feedback is introduced to correct the error. The result of previous prediction is priori, and the measured result is the posteriori. The measurement $\mathbf{z}$ at time k can be obtained from the mapping of system state variables. $\mathbf{H}$ is the transition matrix maps the state variable to the measurement. The random variable $\mathbf{v}$ is the measurement noise.

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k.$$

For a uniform acceleration model, assume the object is uniformly accelerated moving away, the distance of the object displacement is

$$\mathbf{z}_k = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{disp}_k \\ \mathbf{vel}_k \end{bmatrix}.$$

The system noise $\mathbf{w}$ and the measurement noise $\mathbf{v}$ exist in the equation for the state, distribute as multivariate Gaussian distribution, and they are independent of each other.

$$p(\mathbf{w}) \sim N(0, \mathbf{Q}),$$

$$p(\mathbf{v}) \sim N(0, \mathbf{R}),$$

$\mathbf{Q}$ is the covariance of system noise and $\mathbf{R}$ is the covariance of measurement noise. If the system noise exists only on velocity component, the variance of the noise of velocity is constant c, then there's only displacement in measurement. Its covariance

matrix $\mathbf{R}$ is the variance of the measured noise $\mathbf{v}$.

$$\mathbf{R} = E[\mathbf{v}\mathbf{v}^T].$$

In matrix $\mathbf{Q}$, the system noise variance added to velocity component is constant c, and to displacement component is 0. The covariance between velocity and displacement is 0, that is, there is no correlation between noises,

$$\mathbf{Q} = \begin{bmatrix} 0 & 0 \\ 0 & c \end{bmatrix}.$$

The optimal estimate is calculated by weighting (feedback) of the prediction $\mathbf{x}$ (priori) and the measurement $\mathbf{z}$ (posteriori). The notation $\mathbf{x}_{n|m}$ represents the estimate of $\mathbf{x}$ at time n given measurements up to and including at time m.

$$\mathbf{y}_k = \mathbf{z}_k - \mathbf{H}_k\mathbf{x}_{k|k-1},$$

$$\mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_{k|k} - \mathbf{z}_{k|k-1}) = \mathbf{x}_{k|k-1} + \mathbf{K}_k\mathbf{y}_k,$$

$\mathbf{y}_k$ is the measurement pre-fit residual at time k, which is the difference between the prediction and actual measurement. $\mathbf{y} = 0$ means the prediction is exactly the same as the measurement. $\mathbf{x}_{k|k}$ is state estimate, $\mathbf{x}_{k|k-1}$ is the prediction of state (priori), $\mathbf{z}_{k|k}$ is measurement, and $\mathbf{z}_{k|k-1}$ is the prediction of measurement. $\mathbf{K}_k$ is the optimal Kalman gain at time k.

Optimal Kalman gain $\mathbf{K}_k$ can be calculated by the prediction uncertainty $\mathbf{P}_{k|k}$ and the pre-fit residual covariance $\mathbf{S}_k$.

$$\mathbf{S}_k = \mathbf{R}_k + \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T,$$

$\mathbf{P}_{k|k}$ is the error covariance matrix between the estimate and the measurement at time k, which measure the accuracy of the state estimate $\mathbf{x}$. $\mathbf{P}_{k|k-1}$ is the error covariance between the prediction and the measurement, which can be predicted by previous uncertainty with new uncertainty $\mathbf{Q}_k$.

$$\mathbf{P}_{k|k} = E[e_{k|k} - e_{k|k}^T] = E[(\mathbf{x}_k - \mathbf{x}_{k|k})(\mathbf{x}_k - \mathbf{x}_{k|k})^T],$$

$$\mathbf{P}_{k|k-1} = E[e_{k|k} - e_{k|k-1}^T] = E[(\mathbf{x}_k - \mathbf{x}_{k|k})(\mathbf{x}_k - \mathbf{x}_{k|k-1})^T],$$

e_k consists of errors in various system state variables. In the uniform acceleration motion model mentioned above, e_k is a covariance matrix composed of displacement

error and velocity error. Substitute the state estimate $\mathbf{x}_{k|k}$ into $\mathbf{P}_{k|k}$, it becomes

$$\mathbf{P}_{k|k} = E[[(I - \mathbf{K}_k\mathbf{H})(\mathbf{x}_k - \mathbf{x}_{k|k-1}) - \mathbf{K}_k\mathbf{v}_k][(I - \mathbf{K}_k\mathbf{H})(\mathbf{x}_k - \mathbf{x}_{k|k-1}) - \mathbf{K}_k\mathbf{v}_k]^T].$$

System state $\mathbf{x}$ and measurement noise $\mathbf{v}$ are independent of each other, so

$$\mathbf{P}_{k|k} = (I - \mathbf{K}_k\mathbf{H}_k)E[(\mathbf{x}_k - \mathbf{x}_{k|k-1})(\mathbf{x}_k - \mathbf{x}_{k|k-1})^T](I - \mathbf{K}_k\mathbf{H}_k)^T + \mathbf{K}_kE[\mathbf{v}_k\mathbf{v}_k^T]\mathbf{K}_k^T$$

$$= (I - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}(I - \mathbf{K}_k\mathbf{H}_k)^T + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^T.$$

The trace of error covariance matrix $\mathbf{P}$ is the sum of diagonal elements, which contains the variances of the variables.

$$T[\mathbf{P}_{k|k}] = T[\mathbf{P}_{k|k-1}] - 2T[\mathbf{K}_k\mathbf{H}_k\mathbf{P}_{k|k-1}] + T[\mathbf{K}_k(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k)\mathbf{K}_k^T].$$

Minimize $T[\mathbf{P}_{k|k}]$ to get minimum mean square error. Optimal Kalman gain $\mathbf{K}_k$ can be calculated by taking the derivation of $T[\mathbf{P}_{k|k}]$ with respect to $\mathbf{K}_k$ and making the derived function equal to zero.

$$\frac{\mathrm{d}T[\mathbf{P}_{k|k}]}{\mathrm{d}\mathbf{K}_k} = -2(\mathbf{H}_k\mathbf{P}_{k|k-1})^T + 2\mathbf{K}_k(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k),$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T(\mathbf{R}_k + \mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T) = \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{S}_k^{-1}.$$

The transition matrix $\mathbf{H}$ and the covariance of measurement noise R are constant, so the optimal Kalman gain $\mathbf{K}$ is related to the error covariance of the prediction $\mathbf{P}_{k|k-1}$. $\mathbf{K}_k$ increases as the $\mathbf{P}_{k|k}$ increases, and the weight will pay more attention to the feedback. If $\mathbf{P}_{k|k-1}$ equals to zero, the prediction is same as measurement, then $\mathbf{x}_{k|k}$ equals to $\mathbf{x}_{k|k-1}$.

Substitute $\mathbf{K}_k$ into $\mathbf{P}_{k|k}$, the estimation covariance matrix $\mathbf{P}$ becomes

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{P}_{k|k-1}\mathbf{H}_k^T(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^T + \mathbf{R}_k) - 1\mathbf{H}_k\mathbf{P}_{k|k-1}$$

$$= \mathbf{P}_{k|k-1} - \mathbf{K}_k\mathbf{H}_k\mathbf{P}_{k|k-1}$$

$$= (I - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1}.$$

At time k, current estimate covariance matrix $\mathbf{P}_{k|k}$ is calculated by $\mathbf{P}_{k|k-1}$. To calculate new covariance matrix $\mathbf{P}$ at time $k+1$, the $\mathbf{P}_{k|k-1}$ is needed. According

to previous equation, state $\mathbf{x}$ at time $k+1$ is

$$\mathbf{x}_{k+1|k} = \mathbf{A}_{k+1}\mathbf{x}_{k|k} + \mathbf{B}_{k+1}u_{k+1}.$$

Substitute $\mathbf{x}_{k+1|k}$ into $\mathbf{P}_{k+1|k}$, it becomes

$$\mathbf{P}_{k+1|k} = E[e_{k+1|k} - e_{k+1|k}^T] = E[(\mathbf{x}_{k+1} - \mathbf{x}_{k+1|k})(\mathbf{x}_{k+1} - \mathbf{x}_{k+1|k})^T]$$

$$= E[(\mathbf{A}_{k+1}(\mathbf{x}_k - \mathbf{x}_{k|k}) + \mathbf{w}_{k+1})(\mathbf{A}_{k+1}(\mathbf{x}_k - \mathbf{x}_{k|k}) + \mathbf{w}_{k+1})^T]$$

$$= E[(\mathbf{A}_{k+1}e_{k|k})(\mathbf{A}_{k+1}e_{k|k})^T] + E[\mathbf{w}_{k+1}\mathbf{w}_{k+1}^T]$$

$$= \mathbf{A}_{k+1}\mathbf{P}_{k|k}\mathbf{A}_{k+1}^T + \mathbf{Q}_{k+1}.$$

The recursive formula of $\mathbf{P}_{k|k-1}$ is

$$\mathbf{P}_{k|k-1} = \mathbf{A}_k\mathbf{P}_{k-1|k-1}\mathbf{A}_k^T + \mathbf{Q}_k.$$

Thus, for recursive derivation, a initial error covariance matrix $\mathbf{P}_{k|k-1}$ is the only requirement.

In general, the state of Kalman filter can be represented by two variables: $\mathbf{x}_k$, the state estimate at time k; $\mathbf{P}_k$, error covariance matrix at time k, measures the accuracy of the state estimate $\mathbf{x}$. The Kalman filter consists of two steps: Predict and Update.

In predict phase, the filter uses the estimation of previous state $\mathbf{x}_{k-1}$ and external input $\mathbf{u}_k$ to estimate current state $\mathbf{x}_k$.

$$\mathbf{x}_{k|k-1} = \mathbf{A}_k\mathbf{x}_{k-1|k-1} + \mathbf{B}_k\mathbf{u}_k.$$

The new uncertainty $\mathbf{Q}_k$ appears in prediction process, the error covariance $\mathbf{P}_{k|k-1}$ of current state can be calculated by previous uncertainty with new uncertainty.

$$\mathbf{P}_{k|k-1} = \mathbf{A}_k\mathbf{P}_{k-1|k-1}\mathbf{A}_k^T + \mathbf{Q}_k.$$

In update phase, the filter optimizes the prediction $\mathbf{y}_k$ by observed current state $\mathbf{z}_k$ to refine the state estimate $\mathbf{x}_k$. First calculate the measurement pre-fit residual $\mathbf{y}_k$, the pre-fit residual covariance $\mathbf{S}_k$ and the optimal Kalman gain $\mathbf{K}_k$.

$$\mathbf{y}_k = \mathbf{z}_k - \mathbf{H}_k\mathbf{x}_{k|k-1},$$

$$\mathbf{S}_k = \mathbf{R}_k + \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T,$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1}.$$

Update current state estimate $\mathbf{x}_{k|k}$ by weighted averaging the prediction and measurement.

$$\mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + \mathbf{K}_k \mathbf{y}_k.$$

Update current estimate covariance $\mathbf{P}_{k|k}$ and the measurement post-fit residual $\mathbf{y}_{k|k}$ for next iteration.

$$\mathbf{P}_{k|k} = (I - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} (I - \mathbf{K}_k \mathbf{H}_k)^T + \mathbf{K}_k \mathbf{R}_k \mathbf{K}_k^T,$$

$$\mathbf{y}_{k|k} = \mathbf{z}_k - \mathbf{H}_k \mathbf{x}_{k|k}.$$

4.3 Ridge Regression

Ridge regression is a regularization method used in regression analysis of ill-posed problem. A well-posed problem means that: a solution exists; the solution is unique; the solution's behavior changes continuously with the initial conditions. When any of the above conditions are not met, it is called ill-posed problem. For example, if the given three points are not on a straight line, it's not able to get a straight line through the three points at the same time. In other words, the given limitations are too strict, resulting the non-existent solution. In the case where the conditions are not fully satisfied, the linear regression is used to obtain polynomial approximation and fitting regression curve, to find a closest solution.

Ridge regression is one of linear regression. Any problem that can be expressed as $y = Ax + B$ can be regarded as linear regression problem, and a well-posed problem of an unbiased estimation can be expressed as

$$y = \mathbf{A}x,$$

where $\mathbf{A}$ satisfies the full column rank, which is

$$rank(\mathbf{A}) = dim(x).$$

The loss function is the square of the residual, which is the square of the difference between the prediction and the measurement. The smaller the loss, the smaller the residual. So the problem can be seen as a loss function minimization problem

$$min \|\mathbf{y} - \mathbf{A}\mathbf{x}\|^2.$$

The above optimization problem can be solved by

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y}.$$

The current sample cannot provide enough information when $\mathbf{A}$ does not satisfy the full column rank. The determinant of $\mathbf{A}^T\mathbf{A}$ is close to 0, and the error close to infinity. Outliners have significant impact on the results. Therefore, L2 normalization has been added in ridge regression. The loss function becomes

$$\|\mathbf{y} - \mathbf{A}\mathbf{x}\|^2 + \|\alpha\mathbf{I}\mathbf{x}\|^2,$$

and the value of $\mathbf{x}$ can be solved by

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A} + \alpha\mathbf{I})^{-1}\mathbf{A}^T\mathbf{y},$$

where $\mathbf{I}$ is the identity matrix, α is the regularization parameter, $\alpha > 0$. As the α increases, $\mathbf{x}$ becomes smaller and the residual increases. However, smaller α may lead to over-fitting problems.

Ridge regression appropriate to high correlation data, which can eliminate outliers and optimize the model. It enhances the stability of the matrix inversion. In general, ridge regression is used when it is necessary to reduce the variance or the number of features is larger than the number of samples.

5. ASSESSMENT CRITERIA

Assessment criteria is used to evaluate the quality of the model quantitatively. Different metrics are used to evaluate the different aspects of same model, and the choice of metric depends on the type and purpose of the model. This chapter introduces common metrics for multiple object detection and object tracking.

5.1 Object Detection

There are many methods for detection model evaluation, only the accuracy of prediction is not able to assess the whole complex model. In order to accurately evaluate the performance of an object detection model, multiple accuracy metrics are required from different aspects.

5.1.1 Intersection over Union

Object detection needs to locate the bounding box of the object. The bounding box defines the position of the object and separates it from the background. It is necessary that the interest object is as complete as possible in the bounding box and minimize the interference area (occlusion, background, etc).

The bounding box specifies a block of pixels that can be seen as a set of pixels, and the similarity of the set can be described as the accuracy of the detector. Figure 5.1 shows two types of bounding boxes: Green box is the ground truth bounding box, which is manually labeled from the dataset to indicate the location of the object; Red one is predicted bounding box which is the result predicted by the model.

Intersection over union(loU) is used to evaluate the positioning accuracy of the predicted bounding box. loU is the ratio between the intersection and the union of the bounding box predicted by the model and the ground truth bounding box. It is also known as the Jaccard index, used to measure the similarity of the set. Given set A and B, loU is defined as the intersection of the set divided by the union of the

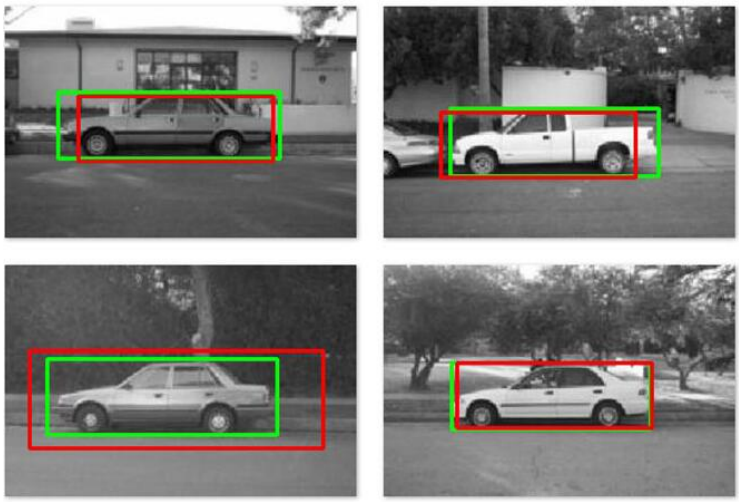


Figure 5.1 *Predicted bounding box and Ground truth bounding box for car.[34]*

set

$$J(A,B) = \frac{|A \bigcap B|}{|A \bigcup B|}$$

As shown in Figure 5.2, the IoU score is the ratio of the overlap area and union area of two bounding boxes. The range of IoU score between 0 and 1. 0 means two bounding boxes are irrelevant, and 1 means they are completely coincident. The more the intersection of predicted bounding box and ground truth bounding box, the higher the IoU score, which indicating that the model predicts the object more accurately. This shows that IoU is an excellent metric for evaluating detectors. Generally, an IoU *score* > 0.5 is normally considered a "good" prediction.

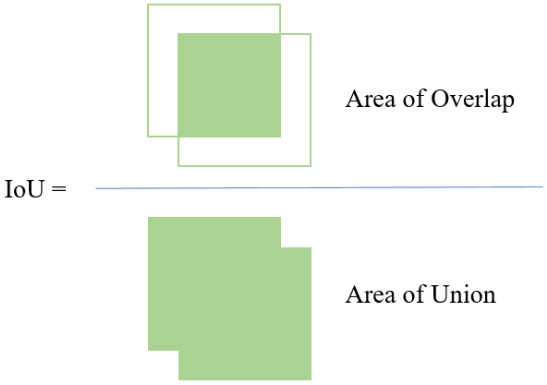


Figure 5.2 *Intersection over union.*

5.1.2 Accuracy Metrics

In object detection, the detection of a single object can be seen as a classification problem. In order to evaluate the ability of the model, we focus on the correctness of the classification, whether the result is correct or not, which is equivalent to a binary classification problem. Figure 5.1 shows the confusion matrix of the binary classifications, and the accuracy can be calculated based on positive and negative classes:

- True Positive (TP), a instance is in positive class and is also predicted as positive, which is, the positive class is correctly predicted.
- True Negative (TN), a instance is in negative class and is also predicted as negative, which is, the negative class is correctly predicted.
- False Positive (FP), wrong prediction (Type I error), the instance is in negative class and is predicted as positive.
- False Negative (FN), missing of the positive prediction (Type II error), the instance is in positive class and is predicted as negative.

Table 5.1 *Confusion matrix.*

		Predicted condition	
		Positive	**Negative**
True condition	**Positive**	**True Positive**	**False Negative**
	Negative	**False Positive**	**True Negative**

Accuracy and Error Rate

Accuracy refers to the ratio of the correctly classified instances and the total number of instances, and is used to measure the ability of the model to correctly predict the class of previously unseen data.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Error rate is the ratio of the incorrectly classified instances and the total number of instances, which indicates the probability of classification error.

$$Errorrate = 1 - Accuracy = \frac{FN + FP}{TP + TN + FP + FN}$$

Precision and Recall

Precision and recall are widely used assessment metrics in machine learning field. It allows us to focus on interest classes to evaluate the quality of the model, and solves the accuracy paradox problem mentioned above.

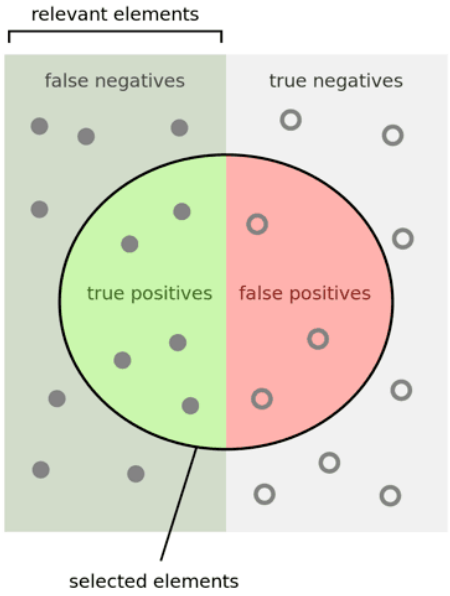


Figure 5.3 *Relationship between true positive, false positive, true negative and false negative.[40]*

Figure 5.3 describes the relationship between the four cases in binary classification. Points represent instances. Solid points on left side are positive class instances and hollow points on right side are negative class instances. The black cycle in middle represent the result predicted by classifier. Both solid and hollow points inside the cycle are predicted as positive, and outside the cycle are predicted as negative.

Precision focuses on prediction phase. It indicates how many true positive instances in instances which are predicted as positive. There are two possible situations when prediction results are positive: predicts the positive instances as positive (TP), and predicts the negative instances as positive (FP). Thus, precision is the proportion of correctly predicted positive instances in all predicted positive instances.

$$Precision = \frac{TP}{TP + FP}$$

Recall focuses on actual instances. It indicates how many true positive instances are correctly predicted. All true positive instances include the true positive instances

which are predicted as positive (TP) and the true positive instances which are predicted as negative (FN). Recall is the proportion of correctly predicted positive instances in all true positive instances.

$$Recall = \frac{TP}{TP + FN}$$

Although the value of both Precision and Recall are expected to be as high as possible, they are contradictory in some cases. When the model is greedy and tries to cover more instances, recall is high enough, but it will lose precision. If the model is conservative and only predicts instances that are very confident, it will has a high precision but low recall. In this way, it is able to focus on the precision and recall of interested class to evaluate the model. Such as cancer detection, which can tolerate low precision, but a high recall is important. In this case, although several cancers are incorrectly reported (low precision), but all true cancers are not missed (high recall).

F1-Score

Due to the contradictory occurring in precision and recall, F-Score is proposed to evaluate the model by combining them comprehensively. F-Score conveys the balance between the precision and the recall, which is the harmonic average of them. The model has perfect precision and recall when F-Score equals 1, and it is the worst when F-Score equals 0. The derivation process is as follows:

$$\frac{2}{F_1} = \frac{1}{precision} + \frac{1}{recall},$$

$$F_1 = (\frac{recall^{-1} + precision^{-1}}{2})^{-1} = 2 \cdot \frac{precision \cdot recall}{precision + recall},$$

$$F_1 = \frac{2TP}{2TP + FP + FN}.$$

Mean Average Precision

In object detection, both the classification and positioning of objects need to be evaluated. Each image may contain multiple objects in different classes and different positions. Therefore, the accuracy metrics used in single-label image classification cannot be directly applied to object detection. There are multiple objects in each class, so the average precision (AP) of all objects in same class needs to be calculated

to evaluate the quality of model in this class. And a mean average precision (mAP), which takes the average of all APs of classes, is needed to evaluate the quality of model in all classes.[19]

AP and mAP are used to evaluate object detection in different challenges, such as PASCAL VOC[9], ImageNet[7], and COCO[26]. Assume that there are M positive samples in N samples for class C, so there are M recall values $r_1, r_2, r_3, \ldots, r_m$. For each recall value r, calculate the maximum precision value p, then M precision values $p_1, p_2, p_3, \ldots, p_m$ are obtained, the AP for class C is the average of all M precision values.

$$AP_C = \frac{\Sigma p_m}{N}.$$

The mAP is the average of all APs, which can be represented as

$$mAP = \frac{\Sigma AP_C}{NumberofClasses}.$$

AP calculates the average precision for each class, measures the performance of the model in single class. mAP takes the average of APs of all classes, measures the general performance of the model for all class.

ROC Curve

The following rates can be calculated from the confusion matrix (true positive, true negative, false positive and false negative) generated by the binary classification problem:

- True positive rate (TPR), also called sensitivity, the recall, indicates the proportion of correctly predicted positive instances in all true positive instances.

$$TPR = \frac{TP}{TP + FN}.$$

- True negative rate (TNR), also known as specificity, is the proportion of correctly predicted negative instances in all true negative instances. All true negative instances include the true negative instances which are predicted as positive (FP) and the true negative instances which are predicted as negative (TN).

$$TNR = \frac{TN}{FP + TN}.$$

- False positive rate (FPR) is the proportion of incorrectly predicted positive instances (they are negative in actual) in all true negative instances, can be calculated as 1−specificity.

$$FPR = \frac{FP}{FP + TN} = 1 - TNR.$$

In binary classification model, assume the threshold is 0.5, the instances with its score higher than threshold will be classified as positive, lower than threshold will be classified as negative. If reduce the threshold to 0.4, the model can predict more positive instances, which increases the value of TPR, but also predicts more true negative instances as positive, which increases the value of FPR.

Receiver Operating Characteristic Curve (ROC curve) also known as sensitivity curve, which plots TPR against FPR at different values of threshold. The horizontal axis indicates FPR and the vertical axis indicates TPR. It can visually represent the trade-off between TPR and FPR, and is used to evaluate the overall quality of the classifier.

In Figure 5.4, the curve shows the relationship between FPR and TPR for different thresholds. There are four extreme situations:

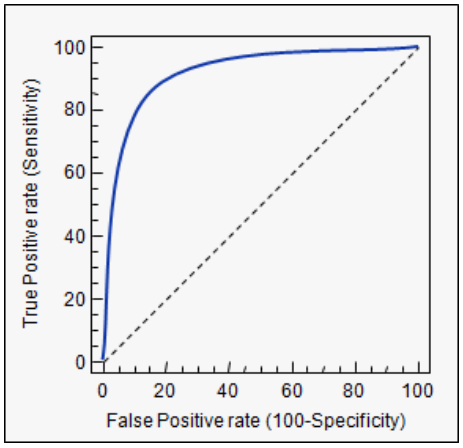


Figure 5.4 *ROC curve.*

- At point $(0, 0)$, both FPR and TPR equal to 0, so that $FP = TP = 0$, which means all instances are predicted as negative by classifier;
- At point $(0, 1)$, where $FPR = 0, TPR = 1$, so that $FN = 0$ and $FP = 0$, all instances are correctly predicted;
- At point $(1, 0)$, where $FPR = 1, TPR = 0$, no correct prediction;

- At point $(1, 1)$, both FPR and TPR equal to 1, all instances are predicted as positive by classifier;

It is obvious that when the ROC curve closer to the upper left corner, the performance of the classifier is better.

P-R curve

Precision-Recall curve (P-R curve) is the curve related to precision and recall. Compared to ROC curve, both of them need TPR(Recall), but ROC curve calculates FPR, P-R curve calculates precision. That is, both indicators of the P-R curve are focused on the positive instances.

Presion and recall have been defined in above section. For creating the P-R curve, the prediction results are sorted in order of highest score to lowest score. Then take each score as the threshold. Instances with scores higher than threshold are predicted as positive, others are negative. Then it is able to calculate TP, FP, and FN, as well as precision and recall. In second turn, take the second score as threshold, and so on.

Table 5.2 *Example data for P-R curve.*

Instance	Class	Score	Instance	Class	Score
1	positive	0.90	11	positive	0.40
2	positive	0.80	12	negative	0.39
3	negative	0.70	13	positive	0.38
4	positive	0.60	14	negative	0.37
5	positive	0.55	15	negative	0.36
6	positive	0.54	16	negative	0.35
7	negative	0.53	17	positive	0.34
8	negative	0.52	18	negative	0.33
9	positive	0.51	19	positive	0.30
10	negative	0.50	20	negative	0.10

Table 5.3 *Part of calculation process.*

Threshold	TP	FP	FN	Precision	Recall
0.90	1	0	9	1	0.1
0.80	2	0	8	1	0.2
0.70	2	1	8	0.67	0.2
0.60	3	1	7	0.75	0.3

Table 5.2 is a list of sample data, and there are 10 positive instances and 10 negative instances in actual. Part of the calculation process is in Table 5.3. The 20 pairs of precision and recall values obtained after all calculations completed can be used to plot P-R curve.

Since the above example data is generated randomly, the resulting P-R curve is not typical enough. Figure 5.5 shows a classic P-R curve which is not related to the above example. The closer the P-R curve is to the upper right corner, the higher the precision and recall, and the better the model performance.

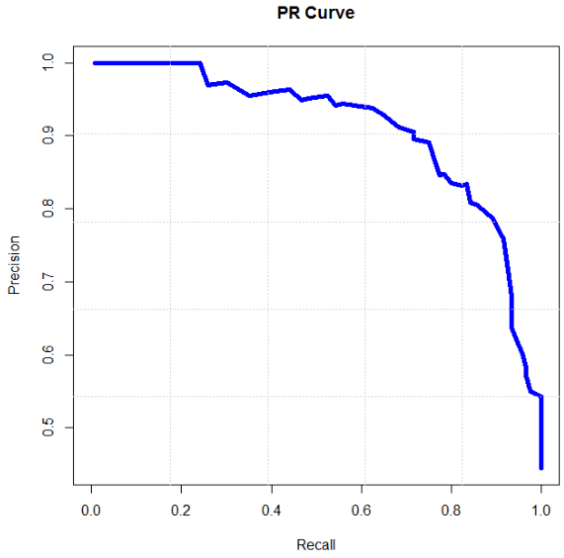


Figure 5.5 *Precision-Recall curve.*

5.1.3 Detection Speed

Object detection also has high requirements for speed. The object detection model is usually inferred from the deep learning framework (Inception[37], VGG[36], Resnet[16]), which has higher computational complexity and requires more memory. Real-time detection is only possible with fast detection speed, which is extremely important for some application scenarios. Frame Per Second (FPS) is widely used to evaluate the detection speed, indicates the number of images that can be processed per second. The higher the FPS, the faster the processing speed.[31]

In addition, the time of processing single image also can be used to evaluate the detection speed. Google published TensorFlow Object Detection API[21] in 2017 and compared the performance of Faster R-CNN, R-FCN and SSD algorithms on MS COCO dataset.

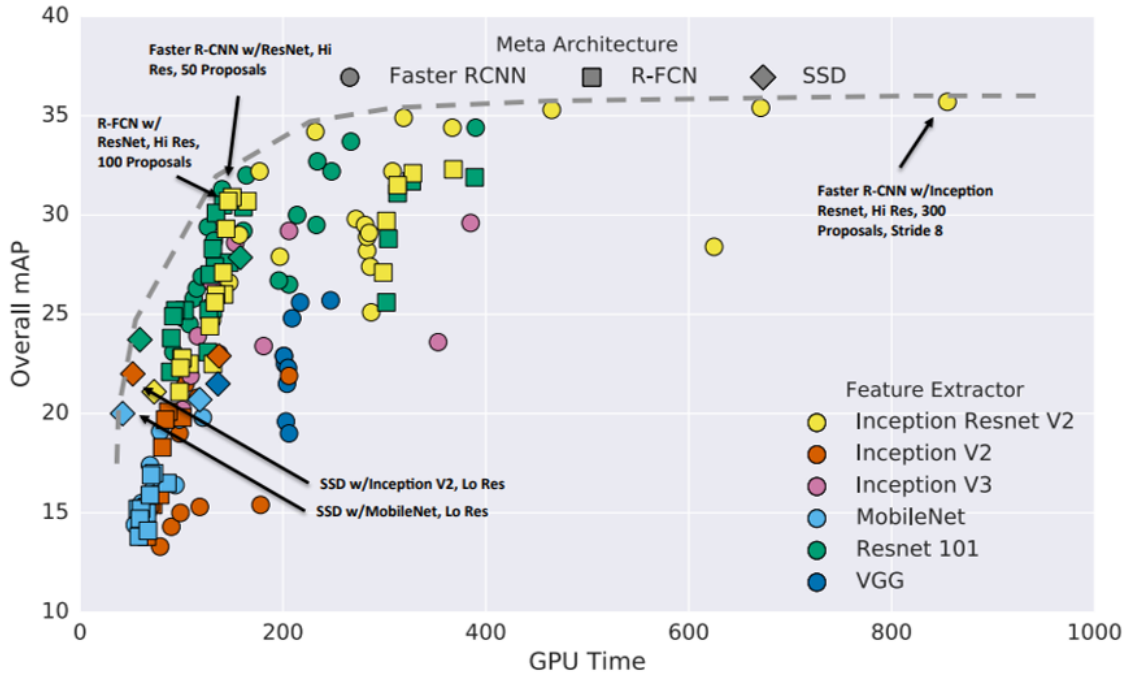


Figure 5.6 *Performance of Faster R-CNN, R-FCN and SSD on MS COCO dataset.*

Figure 5.6 shows that accuracy and speed are a pair of contradictions. High accurate model usually requires more GPU time. In the figure, models with fast detection speeds have lower accuracy, while models with high accuracy require more calculation times. Inception Resnet ($mAP > 35, GPUTime > 800$) has higher accuracy than others, but the detection speed is about 8 times slower than MobileNet ($mAP < 20, GPUTime < 100$). Thus, It is important to consider the trade-off of accuracy and speed of the model when evaluating.

5.2 Multiple Object Tracking

The metrics of multiple object tracking mainly considers the accuracy of the tracker and the completeness of the tracking trajectory. For accuracy and precision, it can be regarded as a binary classification problem, the actual trajectory of the object is the true condition, and the tracking trajectory of the tracker is the predicted condition.[4]

Multiple Object Tracking Accuracy (MOTA) is an metrics to evaluate the accuracy of multiple object tracking, focus on the matching errors for all frames when tracking. The calculation formula is as follow:

$$MOTA = 1 - \frac{FalseNegative + FalsePositive + IDSwitch}{GroundTruth},$$

$FalseNegative$ is the sum of false negatives of all frames, indicating the missing rate. Similarly, $FalsePositive$ is the sum of false positives of all frames, indicating the false positive rate. $IDSwitch$ is the total times of ID switch occurs in all frame, indicating the mismatch rate. The ID switch refers to the times of ID changes for same object in tracking trajectory, usually reflects the stability of tracking. $GroundTruth$ is the sum of the actual number of objects in all frames. Assume that in frame t, there are FN_t false negative instances, FP_t false positive instances, and GT_t instances in total, ID switch occurs $IDSW_t$ times, then the above formula can also be expressed as

$$MOTA = 1 - \frac{\Sigma_t(FN_t + FP_t + IDSW_t)}{\Sigma_t GT_t}.$$

MOTA only evaluates the performance of the tracker when detecting objects and maintaining trajectories, regardless of the accuracy of the object position. MOTA close to 1 reperests better performance of the tracker.

Multiple Object Tracking Precision (MOTP) is a metric to evaluate the position error of multiple object tracking. The calculation formula is as follow:

$$MOTP = \frac{\Sigma_{t,i} d_{t,i}}{\Sigma_t c_t}.$$

In this formula, t is the index of frames, i is the index of tracking object, c is the number of successful matches, and d represents the matching error. The matching error is the distance between the tracking object i and its ground truth, and it is measured by the overlap rate of their bounding boxes. MOTP is used to evaluate the positioning precision of the tracker but not performance.

In addition to the accuracy, the completeness of tracking is also an important aspect. Completeness refers to the completion of tracking the ground truth trajectories, Figure 5.7 shows three common cases:

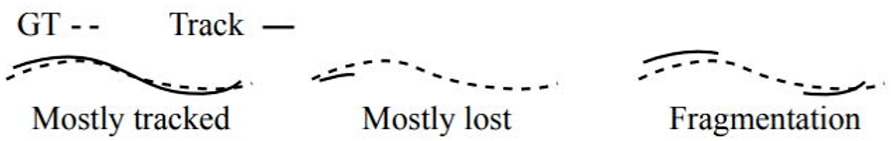


Figure 5.7 *Three cases of relations between predicted tracks and ground truth.*

- Mostly Tracked: Number of tracks that the track matches the ground truth

successfully in more than 80% of time.

- Mostly Lost: Number of tracks that the track matches the ground truth successfully in less than 20% of time.
- Fragmentation: Number of times that the trace is interrupted. Interrupted means the tracker changes to "not tracking" state from "tracking" state suddenly, then back to "tracking" state after a period of time. In other words, the track does not match the ground truth during this period, but it matches before and after the period.

The above cases are not related to the ID switch of the track, but only focus on the matching between track and ground truth.

MOTA and MOTP are more focus on the existence of trajectories, but ID is also an important metric for evaluating the performance of the tracker. The evaluation metrics related to the ID are precision, recall, and F-score of the object ID. The calculation is similar to the object detection mentioned above.

Identification Precision (IDP) refers to the precision of identifying object ID in each bounding box. IDTP is the number of true positive IDs, and IDFP is the number of false positive IDs.

$$IDP = \frac{IDTP}{IDTP + IDFP},$$
$$IDR = \frac{IDTP}{IDTP + IDFN},$$
$$IDF_1 = \frac{2IDTP}{2IDTP + IDFP + IDFN}.$$

IDPrecision, IDRecall and IDF1-score can infer each other based on any two of them. In general, IDF1 is the default metric used to evaluate the quality of the tracker.

6. IMPLEMENTATION

This chapter describes the specific implementation of data preprocessing, object detection, object tracking and tracking trajectory mapping.

6.1 Data Preprocessing

The data preprocessing phase includes the collection of dataset, the annotation and format conversion of training sets and test sets.

Deep learning is very "data hunger". Deep learning learns features from dataset, A large amount of data provides enough materials for training, the larger the dataset, the more features can be covered, which also can avoid over-fitting problem. There are also many parameters in deep learning algorithms need to be fine-tuned, having a large amount of data is the key to generate a more generalized model.

But it is difficult to collect a large dataset. Data collection is very expensive work which costs lot of time. In this case, the deep learning algorithms can be pre-trained from existing large dataset, such as Pascal VOC dataset, COCO dataset, Kitti dataset, etc, then applied and fine-tuned by others, which reduces the impact of the amount of data on model. In this work, the self-collected dataset is small, and deep learning requires large number of training sets, which is not able to train a good performance model directly from our own dataset. So we use pre-trained model and fine-tune the well-trained parameters.

So we use a pre-training model to fine-tune with already trained parameters. Pre-trained models are trained with millions of labelled data, which has powerful generalization capabilities, and less new data is required to produce high performance model. Usually, desired results is able to be obtained after the pre-trained model has been fine-tuned and trained with new dataset.

It is not enough to input data directly into model, the data should be preprocessed to make models easier to find patterns and inference. Well processed dataset can greatly improve the performance of the model. In general, metadata needs to be

added into datasets. The metadata tag used to mark the dataset element is called annotation, which must be accurate and relevant to the tasks. To train the network, large amount of annotated data is required.

The dataset for this project is self-collected from the campus. The video is captured by Rasberry Pi camera, which records the campus squares in different time situations (sunny, cloudy, crowded, etc). We extracted 1438 frames from 9 material videos, and classified the objects into 6 classes: person, cyclist, car, van, bus, truck. Each frame contains multiple objects, 19473 samples has been extracted in total for training. There are more samples of car and person, and fewer samples of truck.

The bounding box annotation method is used in this work. It draws a rectangular box around the bounds of the object so that the whole object and its background are inside the bounding box. These bounding boxes can be used to train the model that uses temporal information to identify, locate, and track objects. The corresponding label should be added to each box to indicate the classification of the object. In this work, we annotate the objects with Imglab tool[22] and store the metadata into xml file.

Imglab is a annotation tool provided by dlib. Dlib is a cross-platform C++ public library which contains many algorithms commonly used in machine learning. Imglab allows you to draw multiple bounding boxes on one image and set the label for each, and it stores the position of box and the label name into extensible markup language (XML) file.

Figure 6.1 *GUI of Imglab.*

Imglab loads all images in folder and shows the image list in graphical interface window, as shown in figure above. The desired object can be marked by drawing the rectangular bounding box around. First input the label name in Next Label field on upper left corner, then press shift button, click and drag left mouse button to draw the bounding box. After the desired objects are fully annotated, click save button in files menu and close the window. The metadata for each boxes will be stored into XML file. The stored information includes the path of the image, and the coordinates of the upper left corner, height and weight, label name of each box. It counts the total number of images and bounding boxes automatically, and the view of XML file is shown in figure below.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<?xml-stylesheet type='text/xsl' href='image_metadata_stylesheet.xsl'?>
<dataset>
<name>imglab dataset</name>
<comment>Created by imglab tool.</comment>
<images>
  <image file='07_30_2018_11_35_25_video/frame_1.jpg'>
    <box top='407' left='456' width='44' height='63'>
      <label>person</label>
    </box>
    <box top='241' left='778' width='6' height='20'>
      <label>person</label>
    </box>
    <box top='287' left='770' width='12' height='30'>
      <label>person</label>
    </box>
    <box top='291' left='777' width='15' height='29'>
      <label>person</label>
    </box>
```

Figure 6.2 *Output .xml file of Imglab.*

Although the Imglab tool is simple and convenient, the annotation for thousands of frames is still a heavy work. In order to speed up and reduce the workload, we uses Faster Bounding Box Annotation method proposed by [2]: annotate a few amount of data and trains a fast SSD model on it, then predicts the unlabelled data to obtain the labelled data. Although the position accuracy of result is not good, but adjust the bounding boxes in result data is much faster than directly annotate the original data, which saves a lot of time.

After the data has been annotated, it is divided into training set and test set, and converted into the format required by the object detection model. We divide all annotated data into 8:2 ratios, 80% as training set and 20% as test set.

Training set is fit on the detection model, which adjusts the parameters of classifier and trains the model. Test set is used to test the accuracy of the trained model. When training the model, the parameters are all fitted by the training set, which may leads to over-fitting problem, and the result of new data predicted by model

will has low accuracy. So it is important to keep the balance of distribution of the classes in both sides when dividing the training set and test set.

In order to fit the data on detection model for training, both training set and test set are converted into TFrecord format. TFrecord is a binary format supported by the model used in thesis, which stores image data and labels together, can be quickly loaded into memory. It does not support random access and is suitable for processing large dataset.

6.2 Object Detection

Object detection with TensorFlow[1] is an open-source machine learning framework released by Google in 2015, which is fast, flexible and suitable for large-scale applications. Main features of TensorFlow are as follows:

- Fast: it is optimized for the hardware and platform, takes full advantage of hardware resources to maximize computing performance;
- Highly portable: it supports various platforms such as Linux, Windows, Mac, and mobile devices;
- Highly flexible: it is able to calculate all kinds of tasks as long as the calculation process can be represented as a data flow graph;
- Plentiful algorithms: nearly all required deep learning algorithms can be found in its library, and it is constantly update new algorithm libraries.

TensorFlow is widely used in machine learning field such as image classification, speech recognition, natural language processing, etc.

Dataflow graphs is used in TensorFlow for numerical calculations. The calculation can be expressed as a directed graph, also known as computation graph. Each node in the graph describes an operation, accepts any number of inputs and outputs. Edge between two nodes represent an association, responsible for the transmission of tensors. Tensor is an N-dimensional vector represents N-dimensional dataset.

The computation graph describes the flow of the data, which looks like the tensor flows along the edges through the nodes in the graph. It is also responsible for maintaining and updating.

The graph must be activated in session for starting the calculation. The session distributes the op of the graph to CPU or GPU device, and provides the pre-defined

function for op. The produced tensor will be returned after the functions are executed.

TensorFlow Object Detection API is used for object detection in this work. It provides pre-trained detection models based on the COCO dataset, the Kitti dataset, the Open Images dataset, the AVA v2.1 dataset and the iNaturalist Species Detection Dataset, and these models have different neural network structures, such as Faster R-CNN, R-FCN, SSD, etc. Meanwhile, an image classification model is needed for object detection to act as Feature Extraction layer. TensorFlow Object Detection API uses Slim library as the feature extractor, which includes several classic convolutional network models, such as AlexNet, VGG16, VGG19, Inception, ResNet, as well as Inception-Resnet and MobileNet published by Google.

In this work, we selected the model which trained on COCO dataset and whose outputs are boxes. Then, there are 5 models which has different networks, suitable speed and mAP has been chosen. The selected models are shown in Table 6.1.

Table 6.1 *Performance of different models trained on COCO dataset.*

Model name	**Speed (ms)**	**COCO mAP**
ssd_mobilenet_v1_coco	30	21
ssd_mobilenet_v2_coco	31	22
ssd_inception_v2_coco	42	24
faster_rcnn_resnet50_coco	89	30
rfcn_resnet101_coco	92	30
faster_rcnn_resnet101_coco	106	32
faster_rcnn_inception_resnet_v2_atrous_coco	620	37

6.3 Multiple Object Tracking

The detection and feature modeling are completed before multiple object tracking. In tracking process, the assignment of object is important. Meanwhile, the creation, reservation, deletion and optimization of tracks should be considered.

The multiple object tracker in this work consist of an assignment algorithm and a set of single object trackers. The video enters into the detection model by frame, and objects in frame are described as bounding boxes. Each object has its own tracker, and the assignment algorithm decides which object belongs to which tracker. The assignment algorithm used is Hungarian algorithm, which calculates the distance between object and all single object trackers. For each single object tracker, it is responsible for tracking specific object by recording the coordinates of the centroid

of the object. The Kalman filter is applied here to obtain a more stable tracking trajectory. Besides, multiple object tracker also achieves the creation, deletion of single object trackers in different conditions, which will be discussed in below.

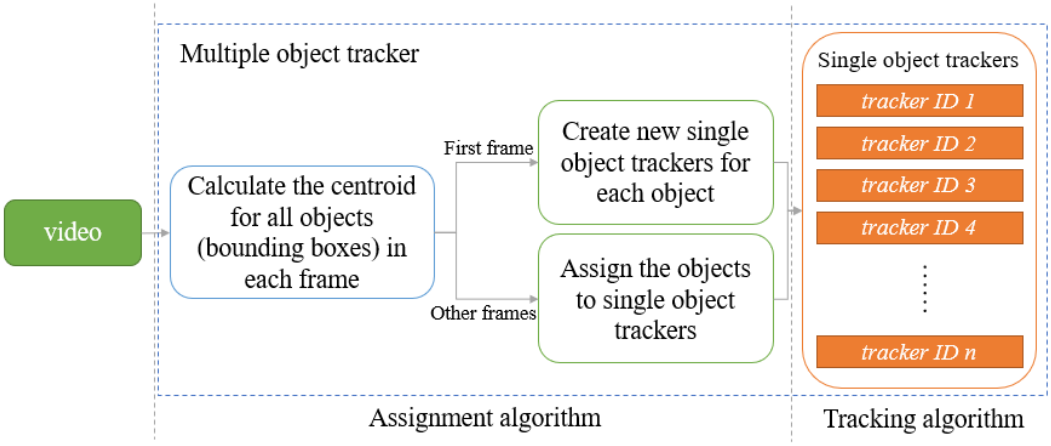


Figure 6.3 *Structure of the multiple object tracker.*

Figure 6.3 shows the structure of the multiple object tracker, which consists of assignment algorithm and tracking algorithm. When the first frame enters into the tracker as input, a new single object tracker with ID is created for each object. For the following frames, we need to match the input objects with the single object trackers, find the matching pair with minimum distance by Hungarian algorithm. Then, the object is used to update the Kalman filter in the matched tracker. The tracker calculates the Kalman gain, state and covariance in Kalman filter, and records the result as newest trajectory of current object.

When the minimum distance is larger than threshold, the object does not match the tracker. There are usually two possibilities for unmatched objects:

- The object is a new one, which has not appeared in previous frames. A new single object tracker should be created for this object.
- The object has appeared in the previous frame, but does not been detected in current frame, which may caused by the occlusion, the moving of the object, and the unstable object detection model. If the object disappears for a long time, it is possible that the object already left the camera range, then the corresponding tracker should be removed. If the disappearance time of object does not exceed the threshold, it may be temporarily occluded by background or other objects. For example, if the pedestrian passes behind the tree, the object detection model cannot provide the bounding box of the person, but its previous trajectory has been recorded in tracker. In this case, the Kalman

filter is able to derive its coordinates of current position based on the previous trajectory, so that it maintains the ID for object and keeps the tracking trajectory continuous.

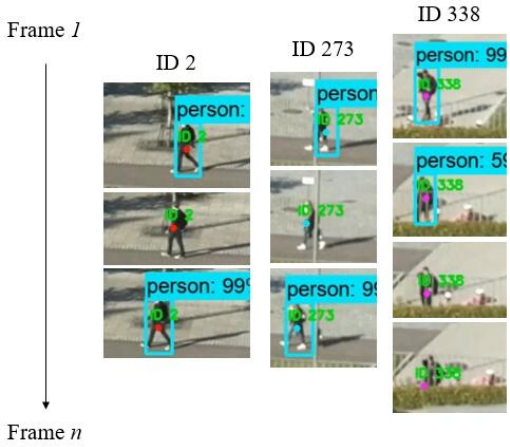


Figure 6.4 *Three cases when there is no detected object in frame.*

In addition, if there is no detected object in frame, the tracker cannot match any object. In this case, the object can be seen as unmatched object, so the calculation and analysis are the same as the above processes. Figure 6.4 shows three cases when object cannot be detected in frame: unstable detection model (ID 2), occlusion (ID 273), moving out of view (ID 338). Kalman Filter Tracking is able to handle the first two cases by predicting the object states, for the third case, the tracker will be kept until the disappearance time of bounding box exceeds the threshold.

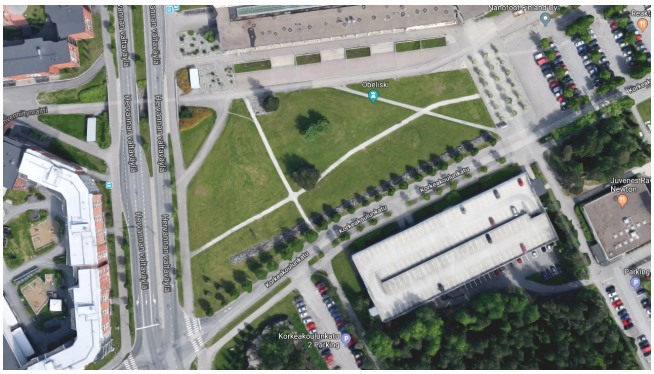
In addition to the Tracker with Kalman Filter Tracking (KF) described above, we also implemented the Tracker with OpenCV Centroid Tracking (CT)[35] for comparison. The CT has the same structure and assignment algorithm as KF, but the tracking algorithm records the centroid of the detected object directly without any prediction.

6.4 Trajectory Mapping

In this work, the video data are collected from the Rasberry Pi camera set on fourth floor of the building. In order to show the result of object tracking more intuitively and conveniently, we expect that the trajectories of the object motion can be drawn from aerial view. Therefore, the result of the object tracking needs to be mapped on on another image which is took from above of the campus square vertically, such as a map. The map is seen as a image, we do not consider the latitude and longitude.

Figure 6.5(a) shows the view of campus square from camera, and Figure 6.5(b) shows the screen-shot of same area from Google map. The camera is placed in the upper right corner of the map, facing the square in the middle of the map. Figure 6.6 describes the position of camera.

(a) The view of campus square from camera.



(b) Screen-shot of campus square from Google map.

Figure 6.5 *Images of the campus square from two different view. The objects in this two images should be matched.*

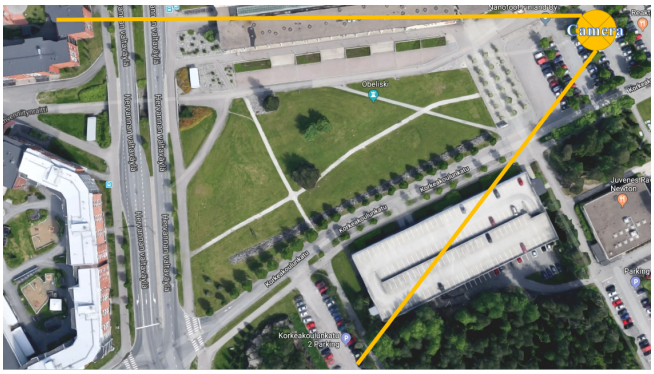


Figure 6.6 *The camera is placed in the upper right corner of the map, facing the square in the middle of the map.*

Trajectory mapping refers to mapping the trajectory of an object motion from one

image to another image. The two images are related to each other. They can be views of the same scene in different angles, so the same object appears in different positions in the two images. The position of the object is described by coordinates, so the mapping function needs to map the pixel coordinates of two images. This can be seen as a regression problem, the corresponding coordinate pair is the data point, the mapping function is treated as a curve, minimize the difference between the curve and the data point to obtain the more suitable mapping function.

In this work, the ridge regression model in Scikit-learn is used to deal with regression problem. Scikit-learn[32] is developed by David Cournapeau in 2007. It is an open source framework developed specifically for machine learning applications in Python language, based on NumPy, SciPy, and Matplotlib packages. Unlike packages as NumPy and Pandas which are able to load, process, and summarize data, Scikit-learn only provides functions related to data modeling: classification, regression, clustering, data dimensionality reduction, feature selection, and feature extraction. Ridge regression model is one of linear regression model in regression function of Scikit-learn.

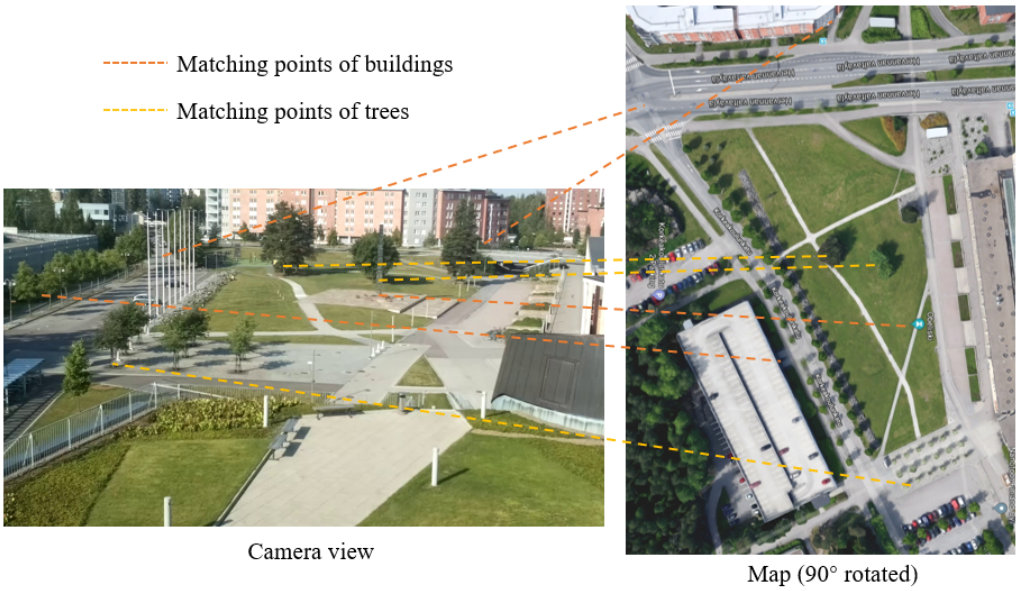


Figure 6.7 *Seven pair of high representative matching points are selected in images. The map is 90 degrees rotated to see the matching more clear.*

We take buildings and trees as reference, and mark 60 points as the training and test data of the ridge regression model. In Figure 6.7 we select only seven pair of high representative matching points and rotate the map in order to show the matching more clear.

7. EVALUATIONS

In this chapter, we evaluate the performance of object detection model based on precision, recall, F1-score, FPS and mAP. For multiple object tracking, we evaluate the performance of the tracker synthetically through precision, recall, F1-score, MOTA and MOTP.

Object Detection

In the experiment, 1150 images containing 15579 objects are trained as training data on fine-tuned models: SSD with MobileNet v1, SSD with MobileNet v2, Faster RCNN with ResNet50, and Faster RCNN with ResNet101. The test dataset contains 288 images with 3894 objects. Table 7.1 describes the accuracy measures of above fine-tuned models. It is obvious that the accuracy of Faster RCNN is higher, but the detection speed is slower, while the detection speed of training and testing SSD is fast, but both precision and recall are low. The performance of SSD with MobileNet v1 is similar to SSD with MobileNet v2, but v2 is slightly better than v1 in all metrics. Although Faster RCNN with ResNet101 has highest recall, the overall performance of ResNet50 is better, which has higher precision and F1-score and faster detection speed. This is because ResNet101 contains more residual layers than ResNet50, which has higher complexity on computation.

Table 7.1 *Accuracy measures of fine-tuned object detection models.*

	Precision	**Recall**	**F1-score**	**FPS**
ssd_mobilenet_v1	0.773	0.579	0.662	17.58
ssd_mobilenet_v2	0.784	0.599	0.673	**17.81**
faster_rcnn_resnet50	**0.940**	0.838	**0.886**	12.96
faster_rcnn_resnet101	0.923	**0.847**	0.884	10.65

Multiple Object Tracking

The results of object detection usually affect the performance of the tracker. To reduce the impact of object detection model on tracker evaluation, we use the dataset provided by *MOTChallenge*[29]. *MOTChallenge* is a framework for fairly evaluating multiple object tracking algorithms, which contains a large number of sequences with detected objects, and provides evaluation tool with a variety of metrics. The provided dataset includes *2D MOT 2015*, *3D MOT 2015*, *MOT 16*, *MOT 17*, *MOT 17 Det* and so on, which are all frame sequences containing annotated objects recorded from different perspectives. In this work, *MOT 17* dataset is used to evaluate and compare the performance between Tracker with Kalman Filter Tracking (KF) and OpenCV Centroid Tracking (CT)[**?**]. *MOT 17* dataset contains ID, position, size and world coordinate system of each bounding box in each frame, focuses on moving pedestrians and vehicles. Since the trajectories of bounding boxes are performed in 2D, the world coordinate system (x, y, z) is -1. Besides, *MOT 17* has more accurate ground truth than *2D MOT 2015* and *MOT 16*. As shown in Figure 7.1, the training set of *MOT 17* contains multiple samples with different number of trajectories and different densities of annotations.

Sample	Name	FPS	Resolution	Length	Tracks	Boxes	Density	Description
	MOT17-13-SDP	25	1920x1080	750 (00:30)	110	11642	15.5	Filmed from a bus on a busy intersection
	MOT17-11-SDP	30	1920x1080	900 (00:30)	75	9436	10.5	Forward moving camera in a busy shopping mall
	MOT17-10-SDP	30	1920x1080	654 (00:22)	57	12839	19.6	A pedestrian scene filmed at night by a moving camera
	MOT17-09-SDP	30	1920x1080	525 (00:18)	26	5325	10.1	A pedestrian street scene filmed from a low angle.
	MOT17-05-SDP	14	640x480	837 (01:00)	133	6917	8.3	Street scene from a moving platform
	MOT17-04-SDP	30	1920x1080	1050 (00:35)	83	47557	45.3	Pedestrian street at night, elevated viewpoint
	MOT17-02-SDP	30	1920x1080	600 (00:20)	62	18581	31.0	People walking around a large square.

Figure 7.1 *Training set of MOT17 Challenge in Multiple Object Tracking Benchmark.*

In addition, *py-motmetrics* library[18] is used to implement the evaluation of multiple object trackers, which provides a variety of metrics, including MOTA, ID switch, Precision, Recall, mostly tracked, mostly lost and number of fragmentations, which

Table 7.2 *Evaluation measures of MOTChallenge and py-motmetrics library.*

Measure	Better	Description
IDF1	higher	The ratio of correctly identified detections over the average number of ground-truth and computed detections.
Rcll	higher	Percentage of detected targets.
Prcn	higher	Percentage of correctly detected targets.
GT	-	Number of ground truth trajectories.
MT	higher	Mostly tracked. Number of objects tracked for at least 80% of lifespan.
PT	-	Partially tracked. Number of objects tracked between 20% and 80% of lifespan.
ML	lower	Mostly lost targets. Number of objects tracked less than 20% of lifespan.
IDs	lower	Total number of identity switches.
FM	lower	Total number of times a trajectory is fragmented.
MOTA	higher	This measure combines three error sources: false positives, missed targets and identity switches.

are described in Table 7.2.

For a sequence randomly selected from *MOT 17* dataset, Table 7.3 shows the comparison between the Tracker with Kalman Filter Tracking (KF) and OpenCV Centroid Tracking (CT). Both of them use Hungarian algorithm as assignment algorithm. CT is independent of consecutive frames, calculates and records the centroid of each object in current frame, but KF is able to predict the next state of the object according to previous position and speed.

Table 7.3 *Accuracy measures of Tracker with Kalman Filter and OpenCV Centriod Tracker.*

	IDF1	Rcll	Prcn	GT	MT	PT	ML	IDs	FM	MOTA
CT	81.0%	68.7%	**98.8%**	60	33	8	19	21	124	67.7%
KF	**87.9%**	**98.6%**	79.4%	60	52	1	7	19	27	**72.9%**

It is obvious that KF is better than CT in overall measures. There are 60 ground truth trajectories in this sequence ($GT = 60$), 32% of them are lost by CT ($ML = 19$), and 87% of them are tracked by KF ($MT = 52$). This is caused by the prediction feature of KF. When the object is occluded, CT stops the moving of the tracker, and KF predicts the object movement according to its previous velocity and direction. When the object appears at another position in a short distance, CT will lost the trajectory of this object and assign a new tracker, but KF already came to the near place according to the prediction, and will match the object again, which makes the trajectory continues. This will lead to high FM and low MOTA in CT.

Trajectory Mapping

Figure 7.2 is the result of the trajectory mapping, which clearly shows how people moving in campus. The relationship between colors and classes are listed in Table 7.4. Green route indicates pedestrians, the figure shows that mostly people walk along the road, and there are also a small number of people on the lawn. Since this aerial view was recorded by Google map a few years ago, it is different from current scene recorded by camera. The establishment of new landmarks in middle of the square leads to the change of the path. The green route from right middle to upper middle is the current path.

Table 7.4 *The relationship between colors and classes in Figure 7.2.*

Color	red	green	blue	yellow	cyan	purple
Class	person	cyclist	car	van	bus	truck

In addition, in the middle bottom of the figure, the blue route shows the trajectory of the vehicle entering and leaving the garage. Since the vehicles moving on the road on the left side of the figure are blocked by three trees in the center of the campus square, and the detection model is not able to detect the occluded object, the blue route is divided into four segments. Moreover, the camera is placed at the upper right position of the map, the farther the distance from camera, the lower the accuracy of the regression model, so the segmented blue route does not match the road well.

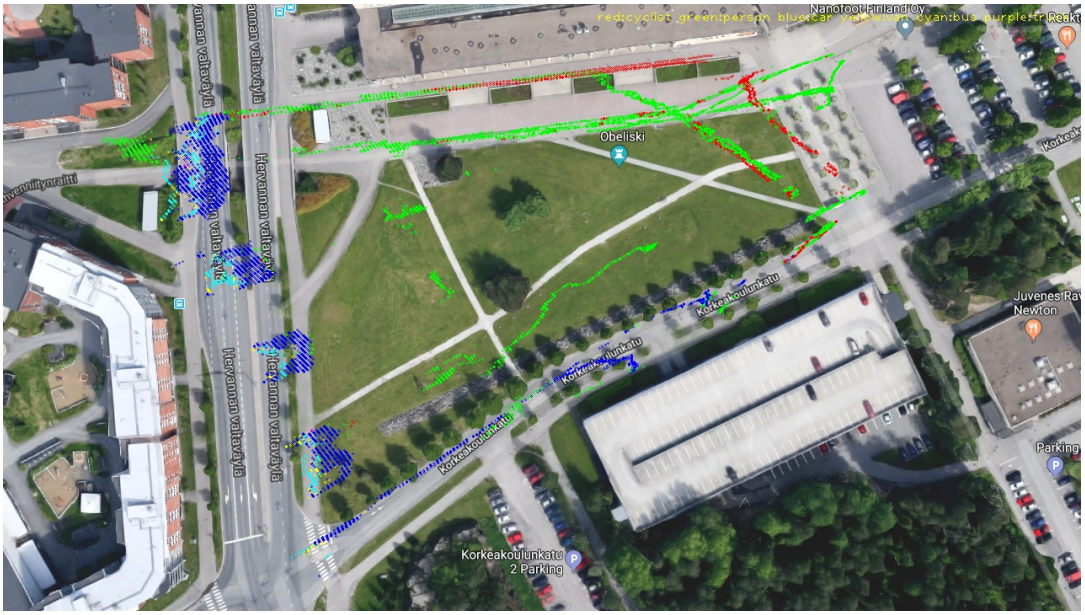


Figure 7.2 *Result of trajectory mapping.*

8. CONCLUSIONS

In this work, several pre-trained models provided by *TensorFlow Object Detection Model Zoo* are used to train self-collected data. The collection of dataset is completed by the capturing the video of campus square by *Rasberry Pi camera* and the annotating the people, cyclists, cars, vans, buses, and trucks in video by *Imglab annotation tool*. For the multiple object tracking, it obtains the objects from detection model, and records the trajectories of all objects in video. After all, a trajectory mapping function is applied to draw the trajectories of moving objects in campus square on campus map.

Two different versions of faster RCNN with ResNet model and SSD with MobileNet model are selected for object detection. From the experiment, we conclude that faster RCNN with ResNet has superior accuracy. Although the SSD model is faster in detection, but the accuracy is lower than others which will influence the tracking process. Both faster RCNN with ResNet50 and ResNet101 have high accuracy for our scenes, but faster RCNN with ResNet50 has better performance considering the computation complexity and the detection speed of the model.

The objects detected by faster RCNN with ResNet50 model is used for multiple object tracking. The test on *MOTChallenge* dataset shows that Tracker with OpenCV Centroid Tracking (CT) and Tracker with Kalman Filter Tracking (KF) have good accuracy and similar position errors. But the ID switches is more frequent in CT, which leads to large number of trajectory fragmentations, and trajectories are mostly lost when the object is occluded. Moreover, KF is able to predict the action according to the previous state of the object. This feature makes KF able to deal with complex situations with superior tracking ability. Although the prediction produces error on object position, the overall performance is still good for tracking.

The resulting trajectory map clearly shows the movement trajectory of pedestrians and vehicles. Although the trajectories far from the camera is not accurate enough, it can be used for statistics on object movements in square or other outdoor environments, see how people or vehicle moves in square and where do they prefer to go. It can be applied to security fields in the future, such as Intelligent Scene Monitoring.

BIBLIOGRAPHY

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," 2016.

[2] B. Adhikari, J. Peltomäki, J. Puura, and H. Huttunen, "Faster bounding box annotation for object detection in indoor scenes," 2018.

[3] E. Alpaydin, *Introduction to Machine Learning*, 2nd ed. The MIT Press, 2010.

[4] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: The clear mot metrics," *EURASIP Journal on Image and Video Processing*, vol. 2008, no. 1, p. 246309, May 2008. [Online]. Available: https://doi.org/10.1155/2008/246309

[5] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theor.*, vol. 13, no. 1, pp. 21–27, Sep. 2006. [Online]. Available: https://doi.org/10.1109/TIT.1967.1053964

[6] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *In CVPR*, 2005, pp. 886–893.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[8] H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik, "Support vector regression machines," in *Proceedings of the 9th International Conference on Neural Information Processing Systems*, ser. NIPS'96. Cambridge, MA, USA: MIT Press, 1996, pp. 155–161. [Online]. Available: http://dl.acm.org/citation.cfm?id=2998981.2999003

[9] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *Int. J. Comput. Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010. [Online]. Available: http://dx.doi.org/10.1007/s11263-009-0275-4

[10] R. Faragher, "Understanding the basis of the kalman filter via a simple and intuitive derivation [lecture notes]," *IEEE Signal Processing Magazine*, vol. 29, no. 5, pp. 128–132, Sep. 2012.

[11] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 9, pp. 1627–1645, Sep. 2010. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2009.167

[12] A. Fuentes, S. Yoon, S. C. Kim, and D. S. Park, "A robust deep-learning-based detector for real-time tomato plant diseases and pests recognition," *Sensors*, vol. 17, no. 9, 2017. [Online]. Available: http://www.mdpi.com/1424-8220/17/9/2022

[13] R. Girshick, "Fast r-cnn," 2015.

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2013.

[15] S. Gould, R. Fulton, and D. Koller, "Decomposing a scene into geometric and semantically consistent regions." in *ICCV*. IEEE Computer Society, 2009, pp. 1–8. [Online]. Available: http://dblp.uni-trier.de/db/conf/iccv/iccv2009.html#GouldFK09

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[17] M. A. Hearst, "Support vector machines," *IEEE Intelligent Systems*, vol. 13, no. 4, pp. 18–28, Jul. 1998. [Online]. Available: http://dx.doi.org/10.1109/5254.708428

[18] C. Heindl, "Benchmark multiple object trackers (mot) in python," https://github.com/cheind/py-motmetrics, 2017.

[19] P. Henderson and V. Ferrari, "End-to-end training of object class detectors for mean average precision," 2016.

[20] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[21] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," 2016.

[22] D. E. King, "Dlib tool imglab," https://github.com/davisking/dlib/tree/master/tools/imglab, 2016.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017. [Online]. Available: http://doi.acm.org/10.1145/3065386

[24] H. W. Kuhn, "Variants of the hungarian method for assignment problems," *Naval Research Logistics Quarterly*, vol. 3, no. 4, pp. 253–258, December 1956. [Online]. Available: https://ideas.repec.org/a/wly/navlog/v3y1956i4p253-258.html

[25] D. D. Lewis, "Naive (bayes) at forty: The independence assumption in information retrieval," in *Proceedings of the 10th European Conference on Machine Learning*, ser. ECML'98. Berlin, Heidelberg: Springer-Verlag, 1998, pp. 4–15. [Online]. Available: https://doi.org/10.1007/BFb0026666

[26] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 740–755.

[27] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," 2015.

[28] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, X. Zhao, and T.-K. Kim, "Multiple object tracking: A literature review," 2014.

[29] A. Milan, L. Leal-Taixe, I. Reid, S. Roth, and K. Schindler, "Mot16: A benchmark for multi-object tracking," 2016.

[30] T. M. Mitchell, *Machine Learning*. McGraw-Hill Education, 1997.

[31] S. Murray, "Real-time multiple object tracking - a study on the importance of speed," 2017.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[33] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2015.

[34] A. Rosebrock, "Intersection over union (iou) for object detection," 2016. [Online]. Available: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

[35] ——, "Simple object tracking with opencv," 2018. [Online]. Available: https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/

[36] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

[38] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, no. 2, pp. 154–171, Sep 2013. [Online]. Available: https://doi.org/10.1007/s11263-013-0620-5

[39] G. Welch and G. Bishop, "An introduction to the kalman filter," Chapel Hill, NC, USA, Tech. Rep., 1995.

[40] Wikipedia contributors, "Precision and recall — Wikipedia, the free encyclopedia," 2019, [Online; accessed 11-April-2019]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Precision_and_recall&oldid=888041025

[41] H. yi Lee, "Deep learning tutorial v3," http://speech.ee.ntu.edu.tw/~tlkagk/talk.html, 2016.

[42] C. Zhang and S. Zhang, *Association Rule Mining: Models and Algorithms*. Berlin, Heidelberg: Springer-Verlag, 2002.